



# FR Family MB2198-01 Emulator System Getting Started Guide

Doc. No. 002-05222 Rev. \*A

Cypress Semiconductor  
198 Champion Court  
San Jose, CA 95134-1709  
<http://www.cypress.com>

**Copyrights**

© Cypress Semiconductor Corporation, 2003-2016. The information contained herein is subject to change without notice. Cypress Semiconductor Corporation assumes no responsibility for the use of any circuitry other than circuitry embodied in a Cypress product. Nor does it convey or imply any license under patent or other rights. Cypress products are not warranted nor intended to be used for medical, life support, life saving, critical control or safety applications, unless pursuant to an express written agreement with Cypress. Furthermore, Cypress does not authorize its products for use as critical components in life-support systems where a malfunction or failure may reasonably be expected to result in significant injury to the user. The inclusion of Cypress products in life-support systems application implies that the manufacturer assumes all risk of such use and in doing so indemnifies Cypress against all charges.

**Trademarks**

All other trademarks or registered trademarks referenced herein are property of the respective corporations.

**Source Code**

Any Source Code (software and/or firmware) is owned by Cypress Semiconductor Corporation (Cypress) and is protected by and subject to worldwide patent protection (United States and foreign), United States copyright laws and international treaty provisions. Cypress hereby grants to licensee a personal, non-exclusive, non-transferable license to copy, use, modify, create derivative works of, and compile the Cypress Source Code and derivative works for the sole purpose of creating custom software and or firmware in support of licensee product to be used only in conjunction with a Cypress integrated circuit as specified in the applicable agreement. Any reproduction, modification, translation, compilation, or representation of this Source Code except as specified above is prohibited without the express written permission of Cypress.

**Disclaimer**

CYPRESS MAKES NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. Cypress reserves the right to make changes without further notice to the materials described herein. Cypress does not assume any liability arising out of the application or use of any product or circuit described herein. Cypress does not authorize its products for use as critical components in life-support systems where a malfunction or failure may reasonably be expected to result in significant injury to the user. The inclusion of Cypress' product in a life-support systems application implies that the manufacturer assumes all risk of such use and in doing so indemnifies Cypress against all charges.

Use may be limited by and subject to the applicable Cypress software license agreement.

# Contents



<b>1. Introduction</b>	<b>6</b>
<b>2. Sample Program</b>	<b>7</b>
2.1 Start new project with Softune Workbench	7
2.2 "Main.c"	7
2.3 Compiling "Main.c"	8
<b>3. Debugging, First Steps</b>	<b>9</b>
3.1 Setup Hardware	9
3.2 Entering Debugger Mode	9
3.3 Using Bookmarks	11
3.4 Start Execution	12
3.5 Stop Execution	12
3.6 Reset MCU	12
<b>4. Monitoring and Manipulating</b>	<b>13</b>
4.1 Monitoring and Manipulating Processor Status	13
4.2 Monitoring and Manipulating CPU Registers	13
4.3 Monitoring and Manipulating Assembly Variables	15
4.4 Monitoring and Manipulating C Variables	16
4.5 Monitoring and Manipulating Memory	17
4.6 Symbol view	19
4.7 Local variables	20
<b>5. Breakpoints</b>	<b>21</b>
5.1 Setting Break points	21
5.1.1 Setting code break point through editor window	21
5.1.2 Setting code break point using Dialog box	23
5.2 Position of Break point	25
<b>6. Events</b>	<b>28</b>
6.1 How to Set Code and Data Events	28
6.1.1 Code Event	29
6.1.2 Data Event	32

<b>7. Trace</b> .....	<b>38</b>
7.1 Trace Window.....	38
7.2 Trace View.....	40
7.3 Trace Jump.....	41
7.4 Back Trace .....	43
7.5 Search Trace .....	44
7.6 Trace Setup .....	45
7.7 Saving Trace Data .....	47
<b>8. Time Measurement</b> .....	<b>48</b>
<b>9. Call Stack</b> .....	<b>50</b>
<b>10. Function Call</b> .....	<b>51</b>
<b>11. Vector</b> .....	<b>53</b>
11.1 Display and setting vectors .....	53
11.1.1 Display .....	53
11.1.2 Setting an address .....	54
11.2 Jump.....	55
<b>12. Debug Environment Setup Procedure</b> .....	<b>56</b>
12.1 Execution.....	56
12.2 Watch .....	57
12.3 Radix .....	58
12.4 Emulation.....	58
12.5 Breakpoint .....	59
12.6 Monitoring.....	60
12.7 Directory .....	61
12.8 Tab .....	61
12.9 Error output.....	62
12.10 Access Size .....	63
12.11 Load.....	64
12.12 External Memory Emulation.....	65
12.13 Frequency.....	66
12.14 Inaccessible area.....	66
<b>13. Trigger-Input and Emulator-Output</b> .....	<b>68</b>
13.1 The BNC Connectors.....	68
13.2 Trigger-Input .....	68
13.3 Emulator-Output .....	69
<b>14. Installing LAN</b> .....	<b>70</b>
14.1 Overview.....	70
14.2 Configuring the LAN Adapter .....	70
14.3 Configuring Operating System “Windows™” .....	71
14.4 Checking the network-connection.....	74
14.5 Troubleshooting .....	74
14.6 Softune Workbench .....	76



<b>15. Miscellaneous</b> .....	<b>77</b>
15.1 View Mode of the Editor.....	77
<b>16. Appendix</b> .....	<b>78</b>
16.1 Related Documents .....	78
<b>17. Additional Information</b> .....	<b>79</b>
<b>Revision History</b> .....	<b>80</b>
Document Revision History .....	80

# 1. Introduction



This document will help you how to debug an emulation system with the MB2198-01 Emulation with the Softune Workbench V60L06. For in-depth information please refer to the following manuals:

- MB2198-01 Hardware Manual (Emulator)
- MB2198-01 Getting Started Application Note (MCU-AN-391027)
- MB2198-01 Installation Guide Application Note (MCU-AN-391026)

This document describes the debugging methods of a MB91V460 system together with a SK-91F467-Flexray target board. Please note, that the debugging principle is the same for other systems.

## 2. Sample Program



Sample Program for Debugging

### 2.1 Start new project with Softune Workbench

At first choose an evaluation MCU (here: MB91467D), copy the template project of the “Softune samples” into an own folder (here: “Emulation\_Test”) and start the Softune Workbench Software.

### 2.2 “Main.c”

The following sample code program, based on the standard template project, is used for demonstrating emulation and debugging. Please change “Main.c” to the following:

```
/******@INCLUDE_START******/
#include "mb91467d.h"
#include "vectors.h"
/******@INCLUDE_END******/

/******@FUNCTION_DECLARATION_START******/
void wait (short int cnt)
{
    int i;
    PDR16 = 0xFF;
    for(i=0;i<cnt;i++);
    PDR16 = 0x00;
}

void main(void)
{
    __EI();                /* enable interrupts */
    __set_il(31);         /* allow all levels */
    InitIrqLevels();     /* init interrupts */

    PORTEN = 0x3;        /* enable I/O Ports */
                        /* This feature is not supported by MB91U460A */
                        /* For all other devices the I/O Ports must be enabled*/

    PFR16 = 0x00;
    DDR16 = 0xFF;


    while(1)             /* endless loop */
    {
        HWWO_CL = 0;
        __asm(" NOP");
        wait (5000);
        __asm(" NOP");
    }
}
```

This program is only an example with no “great assignment“. It contains a simple wait-function (`void wait`), which needs an short integer value for the wait time. The resulting delay time depends on the value itself and the clock speed of the emulation system.

At first the interrupts are enabled (although they are not used in this example), then the Port16 of the MCU is set to “output“ (Port16 is the LED-Port of the SK-91F467-Flexray board). Then the `wait` function is called with the value 5000.

## 2.3 Compiling “Main.c”

To compile the project, please use “Setup Project” first. In `Project`→`Setup Project`→`C Compiler`→`Category`: Optimize has to be selected `General-purpose Optimization Level`: **None**.

Then compile the project by clicking on  selecting `Project`→`Build`, or pressing “Ctrl-F8”. Build all source files regardless of data or

Watch for error messages. If all is ok, you will get the following message:

```

Now building...
-----Configuration: 91460_template_91467D.prj - STANDALONE-----
vectors.c
Start91460.asm
mb91467d.asm
MAIN.c
Now linking...
<your path>91460_template_91467d\STANDALONE\ABS\91460_template_91467d.abs
Now starting load module converter...
<your path>\91460_template_91467d\STANDALONE\ABS\91460_template_91467d.mhx

-----
No Error.
-----

```



# 3. Debugging, First Steps



How to Enter Debugging Mode

## 3.1 Setup Hardware

For the next steps you have to set up your emulation hardware. Please refer to the application notes "Installation Guide MB2198-01" (MCU-AN-391026) and "Emulator System MB2198-01, Getting started" (MCU-AN-391027) for details.

## 3.2 Entering Debugger Mode

After successful compilation of the project start the debugging mode via COM1/2, USB or LAN by double clicking on the regarding Debug-".sup"-entry in the workspace window. After successful connection to the emulator, reset MCU, open "Main.c" (close it first, if it is open) and then click on right mouse button and select "*Mix Display*". Your generated code should look like the following sample code.

```

/*****@INCLUDE_START*****/
#include "mb91467d.h"
#include "vectors.h"
/*****@INCLUDE_END*****/

/*****@FUNCTION_DECLARATION_START*****/
void wait (short int cnt)
000402A0: 1704          ST      R4,@-R15
000402A2: 1781          ST      RP,@-R15
000402A4: 0F02          ENTER   #008
{
    int i;
    PDR16 = 0xFF;
000402A6: CFF0          LDI:8   #FF,R0
000402A8: 8B0D          MOV     R0,R13
000402AA: 1A10          DMOVB  R13,@010
    for(i=0;i<cnt;i++);
000402AC: C000          LDI:8   #00,R0
000402AE: 3FF0          ST      R0,@(R14,-4)
000402B0: 4050          LDUH   @(R14,10),R0
000402B2: 97A0          EXTSH  R0
000402B4: 2FF1          LD      @(R14,-4),R1
000402B6: AA01          CMP     R0,R1
000402B8: EB04          BGE    000402C2
000402BA: 2FF0          LD      @(R14,-4),R0
000402BC: A410          ADD     #1,R0
000402BE: 3FF0          ST      R0,@(R14,-4)
000402C0: EOF7          BRA    000402B0
    PDR16 =0x00;
000402C2: C000          LDI:8   #00,R0
000402C4: 8B0D          MOV     R0,R13
000402C6: 1A10          DMOVB  R13,@010
}
000402C8: 9F90          LEAVE
000402CA: 0781          LD      @R15+,RP
000402CC: A301          ADDSP  #4
000402CE: 9720          RET

void main(void)
000402D0: 1781          ST      RP,@-R15

```

```

-> 000402D2: 0F01          ENTER    #004
    {
-> 000402D4: 9310          ORCCR   #10      /* enable interrupts */
-> 000402D6: 871F          STILM   #1F      /* allow all levels */
    InitIrqLevels(); /* init interrupts */
-> 000402D8: 9F8C00040000 LDI:32  #00040000,R12
-> 000402DE: 971C          CALL    @R12

    PORTEN = 0x3; /* enable I/O Ports */
-> 000402E0: C030          LDI:8   #03,R0
-> 000402E2: 9F8C00000498 LDI:32  #00000498,R12
-> 000402E8: 16C0          STB     R0,@R12
    /* This feature is not supported by MB91V460A */
    /* For all other devices the I/O Ports must be enabled*/

    PFR16 = 0x00;
-> 000402EA: C000          LDI:8   #00,R0
-> 000402EC: 9F8C00000D90 LDI:32  #00000D90,R12
-> 000402F2: 16C0          STB     R0,@R12

    DDR16 = 0xFF;
-> 000402F4: CFF0          LDI:8   #FF,R0
-> 000402F6: 9F8C00000D50 LDI:32  #00000D50,R12
-> 000402FC: 16C0          STB     R0,@R12

    while(1) /* endless loop */
    {
        HWWO_CL = 0;
-> 000402FE: 9F80000004C7 LDI:32  #000004C7,R0
-> 00040304: 8070          BANDL  #7,@R0
        __asm(" NOP");
-> 00040306: 9FA0          NOP
        wait (5000);
-> 00040308: 9B041388      LDI:20  #01388,R4
-> 0004030C: D7C9          CALL    \wait
        __asm(" NOP");
-> 0004030E: 9FA0          NOP
-> 00040310: EOF6          BRA     000402FE
    }
-> 00040312: 9F90          LEAVE
-> 00040314: 0081          LD     @(R13,R8),R1
-> 00040316: 9720          RET


    /*****@FUNCTION_DECLARATION_END*****/
    
```

### 3.3 Using Bookmarks

Since Softune version V60L06 it is possible to set bookmarks in source code windows.


Bookmarked lines are marked with completely in green in source code lines. With the bookmark arrow buttons it can be stepped through the code, stopping at bookmarked lines.

### 3.4 Start Execution

To enter the run mode, click on  **Run continuously** or select *Debug*→*Run*→*Go*, or press “F5”.

Now the program is being executed. If you are using a target system with LEDs on Port16, you will see the LEDs flicker if SK-91F467D-Flexray Target board is used.

### 3.5 Stop Execution

To stop the MCU, click on  **Stop execution** or select *Debug*→*Abort*.

Now the system is halted, but it can be continued again by clicking on “*Run continuously*” or selecting “*Go*”.

### 3.6 Reset MCU

To reset the MCU, click on  **Reset MCU** or select *Debug*→*Reset of MCU*.

Note: This works only if the application is stopped (e.g. breakpoint, etc.)

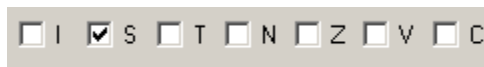
# 4. Monitoring and Manipulating



How to Monitor and Manipulate CPU Registers, Variables and Memory

## 4.1 Monitoring and Manipulating Processor Status

The Condition Code Register (CCR) is always displayed below the workspace window.



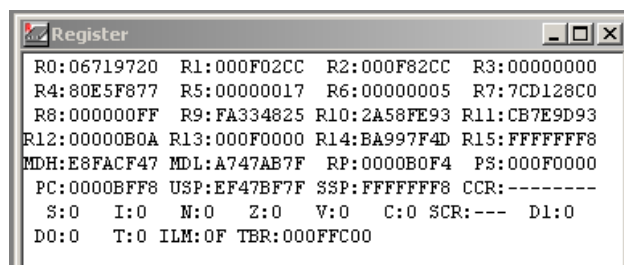
The flags are:

Abbr.	Flag Names
I	Interrupt enable flag (1 = enable)
S	Stack flag (0 = User stack; 1 = System stack)
T	Sticky bit flag (1 = shift right instruction executed)
N	Negative flag (MSB = 1 in last operation)
Z	Zero flag (Last operation resulted in "0")
V	Overflow flag (Overflow at last operation)
C	Carry flag (Last operation caused carry)

The value of the flags can be easily changed by clicking into the white square. A "check mark" (√) indicates that the flag is set (== 1).

## 4.2 Monitoring and Manipulating CPU Registers

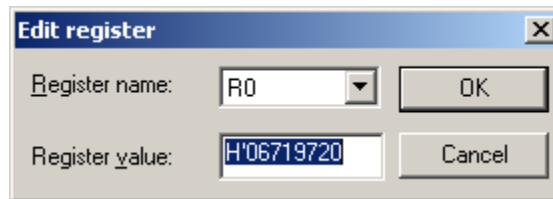
To display the CPU Registers window choose in the debugging mode: *View -> Register*. A new window will occur and look like this:



The registers are:

Abbr.	Flag Names
R0-R12	General purpose registers
R13	Virtual Accumulator
R14	Frame pointer
R15	Stack pointer
PC	Program counter
PS	Program status
TBR	Time Base register
RP	Return pointer
SSP	System stack pointer
USP	User stack pointer
MDH, MDL	Multiply-Divide register
TBR	Interrupt vector Table base register

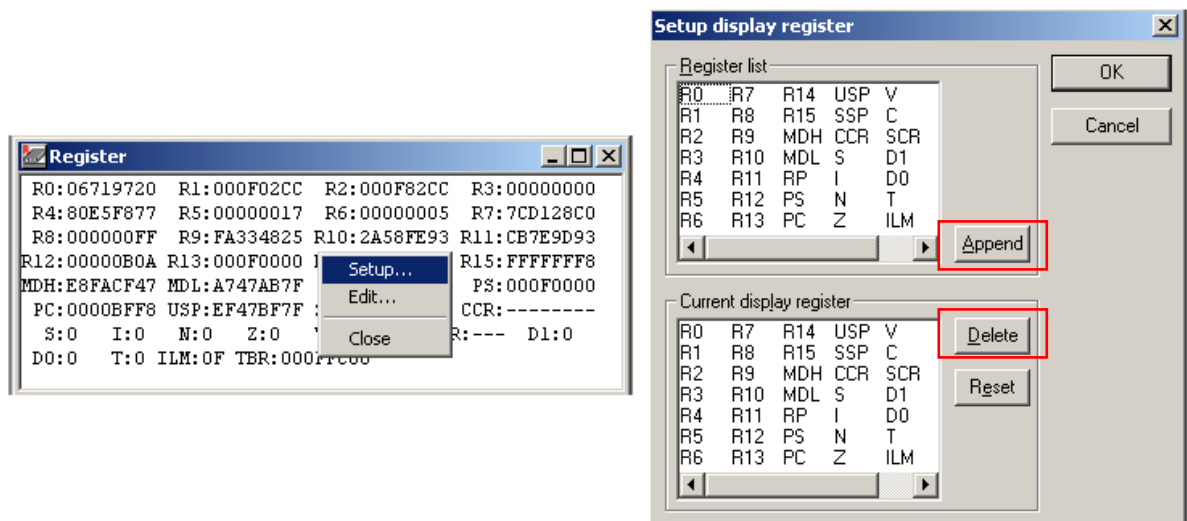
The contents of these registers can be changed by double-clicking them. A pop-up window will occur and look like the following picture:



Under *Register value* option one can enter a new value for the register. Note, that the values always are shown in hexadecimal notation by set-up default, but one can enter even decimal values (beginning with "D"), binary values (beginning with "B"), or octal values

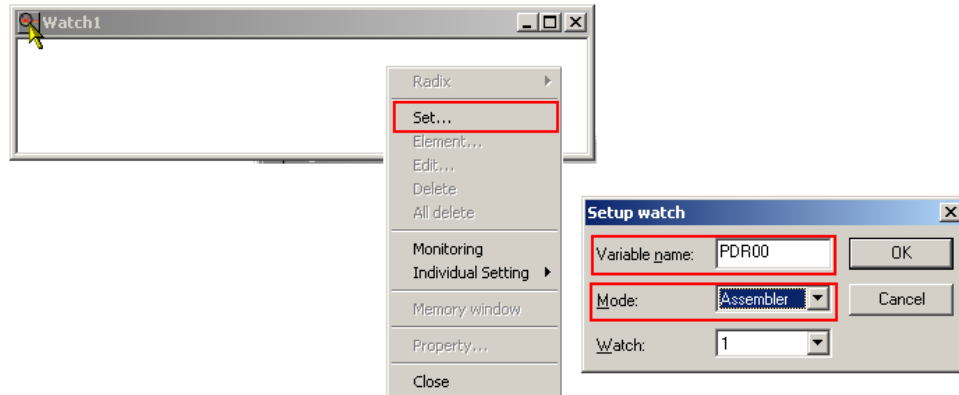
(beginning with "O").

Registers can be added or removed by right clicking on *Register* window and selecting *setup...*

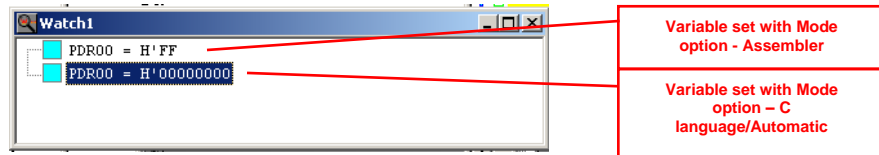


### 4.3 Monitoring and Manipulating Assembly Variables

To display assembly variables choose in the debugging mode: *View -> Watch -> Watch1*. A new window *Watch* will occur. Click in this window on the right mouse button and select *Set...*. A pop-up window *Setup watch* will appear.



Under *Variable name* option one can enter the variable name of the assembly program. The Mode must be *Assembler* in this case. The Watch window will then contain the variable name and value. If we select other than this then watch window will show variables memory location and not the data contained at that location.



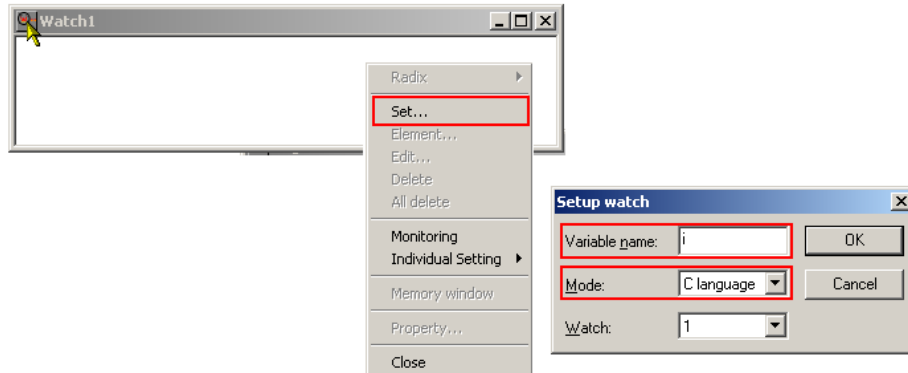
**Note:** You can change the radix of the value by right-clicking on the variable entry and choose via Radix: Binary, Octal, Decimal, or Hexadecimal.

To manipulate the value just double-click on the entry and enter in the pop-up window *Edit variable* a new value. The radix can be chosen via “D”, “H”, “B”, or “O”.

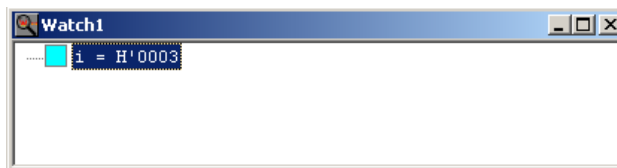


## 4.4 Monitoring and Manipulating C Variables

To display C variables choose in the debugging mode: *View -> Watch -> Watch1*. A new window *Watch* will occur. Click in this window on the right mouse button and select *Set...*. A pop-up window *Setup watch* will appear.

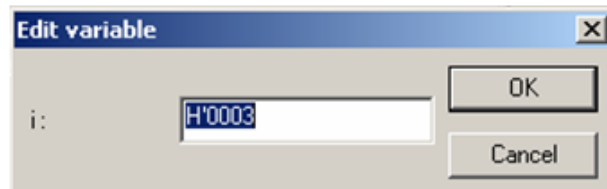


Under *Variable name* option one can enter the variable name of the C program. The Mode must be *C language* or *Automatic* in this case. The Watch window will then contain the variable name and value.



**Note:** You can change the radix of the value by right-clicking on the variable entry and choose via Radix: Binary, Octal, Decimal, or Hexadecimal.

To manipulate the value just double-click on the entry and enter in the pop-up window *Edit variable* a new value. The radix can be chosen via "D", "H", "B", or "O".



As explained in previous chapter to view memory content of Special Function Register, one should select *Assembler* under *Mode* option.

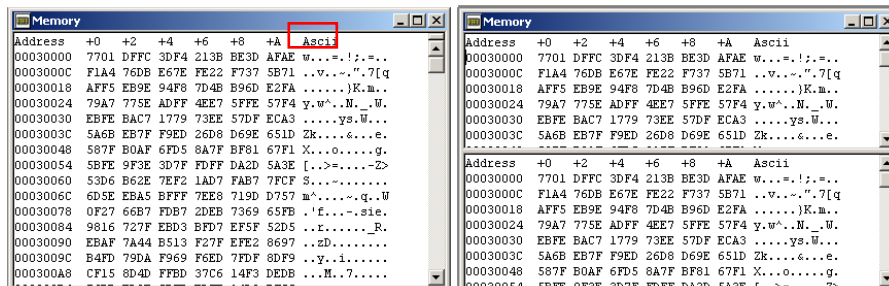


## 4.5 Monitoring and Manipulating Memory

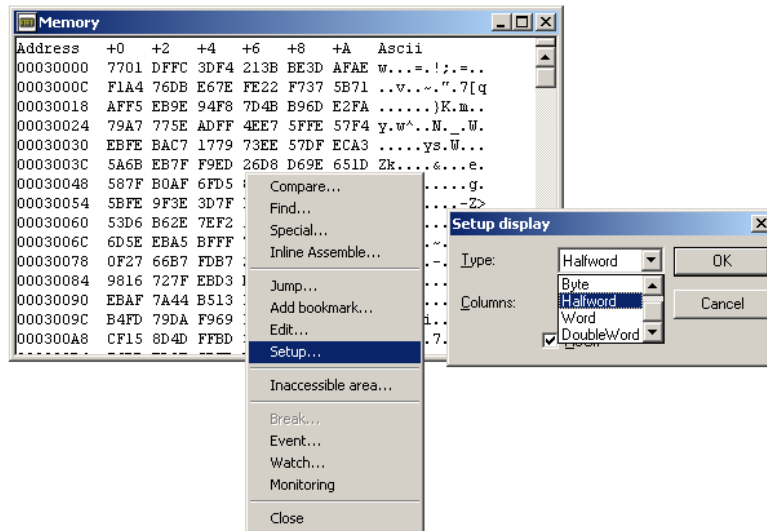
To display the MCU memory choose in the debugging mode: *View -> Memory*. A pop-up window *Memory* will occur and ask for the start address to be displayed. Type for instance H'2530 (or just 2530) for the RAM area. Following window occur which is something like a "Hex-Editor":

Address	+0	+2	+4	+6	+8	+A	Ascii
00030000	7701	DFFC	3DF4	213B	BE3D	AFAE	w...=.!;.=..
0003000C	F1A4	76DB	E67E	FE22	F737	5B71	..v...~".7[q
00030018	AFF5	EB9E	94F8	7D4B	B96D	E2FA	.....)K.m..
00030024	79A7	775E	ADFF	4EE7	5FFE	57F4	y.w^.N_.W.
00030030	EBFE	BAC7	1779	73EE	57DF	ECA3	.....ys.W...
0003003C	5A6B	EB7F	F9ED	26D8	D69E	651D	Zk....&...e.
00030048	587F	BOAF	6FD5	8A7F	BF81	67F1	X...o.....g.
00030054	5BFE	9F3E	3D7F	FDFD	DA2D	5A3E	[...>=....-Z>
00030060	53D6	B62E	7EF2	1AD7	FAB7	7FCF	S...~.....
0003006C	6D5E	EBA5	BFFF	7EE8	719D	D757	m^.....~.q..W
00030078	0F27	66B7	FDB7	2DEB	7369	65FB	.'f...-.sie.
00030084	9816	727F	EBD3	BFD7	EF5F	52D5	..r....._R.
00030090	EBAF	7A44	B513	F27F	EFE2	8697	..zD.....
0003009C	B4FD	79DA	F969	F6ED	7FDF	8DF9	..y..i.....
000300A8	CF15	8D4D	FFBD	37C6	14F3	DEDB	...M..7.....

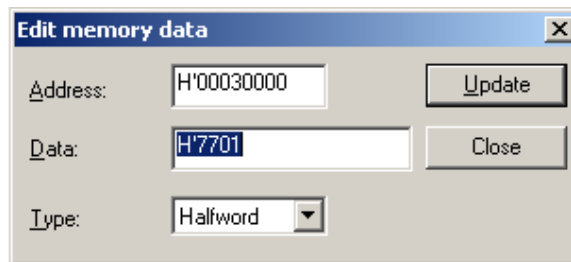
Memory window can be split. Move pointer near to top right corner, pointer will change to ; drag it down with right mouse button pressed to split.



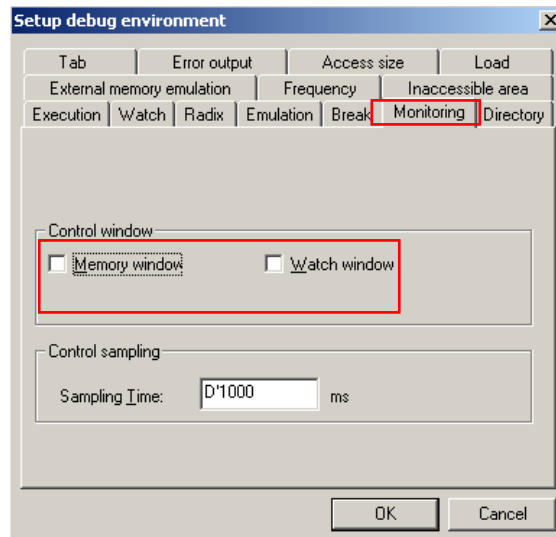
Also view can be set up to see data in bit, byte, word or long. Right click on memory window, click on setup. On a *Setup Display* dialog box select bit, byte, word or long from drop down menu.



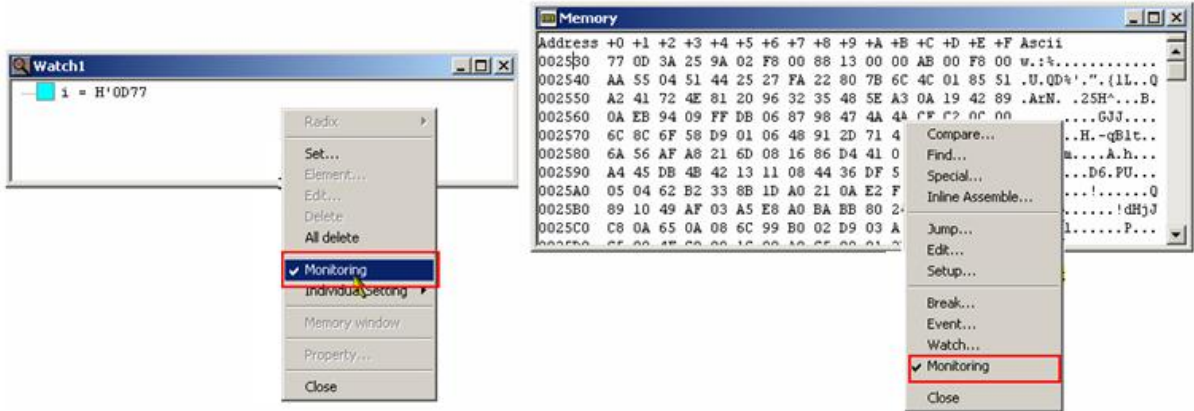
To change a memory content just double click on the respecting byte and *Edit memory data* dialog window pops up. In this window one can specify the address (default is the address of the clicked byte) and the new value. The value can be entered in hexadecimal, decimal, binary or octal format.



To see the memory in “real time” during execution, choose *Setup -> Debug environment -> Debug environment...* Then select the *Monitoring* tab and enter “D'1000” at *Sampling Time* and under *Control window* option click on check box *Memory window* and *Watch window*



Now, when the program is executed, *Watch* and *Memory* window is updated at 1000 ms, and one can see the addresses H'2530 in *Memory* window and variable 'i' in *Watch* window changing its values. Alternately one can select the same functionality by right clicking on *Watch/Memory* window, and on popup menu clicking on *Monitoring*

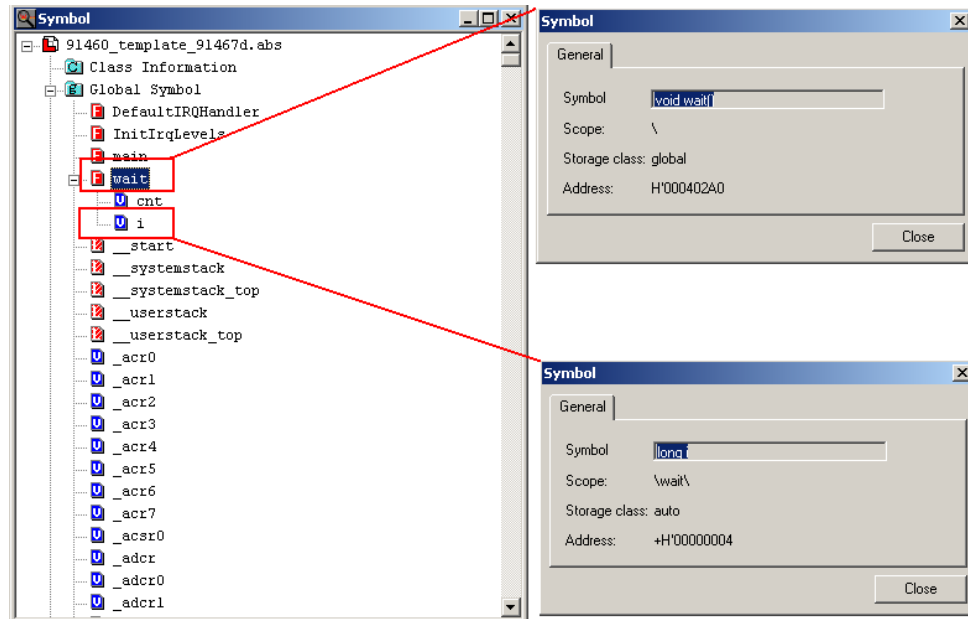


## 4.6 Symbol view

If one want to know where a variable is located in the memory than one can choose




*View -> Symbol*. Then unfold the sub list *Project\_name.abs/Global Symbol*. Click with the right mouse button to the variable you want to get information about and select *Property...*

The Symbol sub window shows then the address:



Icon Reference

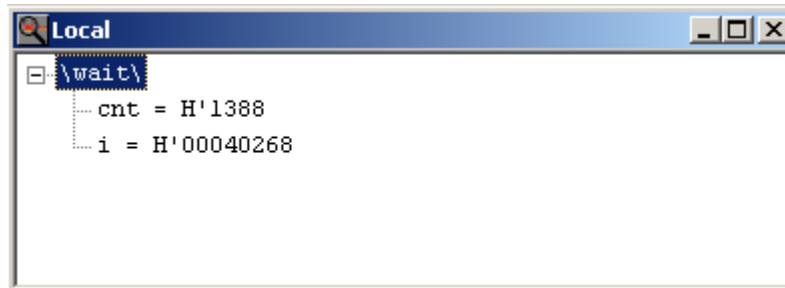
The following icons are used:

Icons	Flag Names
	Function
	Variable
	Label

## 4.7 Local variables

Local variables of functions can be displayed via View ->Local. A new window will open.

Note, that this window only shows contents if the debugger is in stop mode (e.g. breakpoint reached) and the actual function has local variables.



# 5. Breakpoints



## How to Set Break Points

### Code Break Point:

When code break point is set, program execution stops when the PC passes the break address (when instruction at that address is executed).

Four hardware code break points can be set and 4096 software break points can be set. For software break point, program halts every time when PC passes through the set address.

## 5.1 Setting Break points

### 5.1.1 Setting code break point through editor window

Each assembler line in the mixed mode display of the source code has a blue arrow and a green circle symbol

```
12: void wait (short int cnt)
000402A0: 1704          ST      R4,@-R15
000402A2: 1781          ST      RP,@-R15
000402A4: 0F02          ENTER  #008
13: {
14:          int i;
15:          PDR16 = 0xFF;
000402A6: CFF0          LDI:8   #FF,R0
000402A8: 8B0D          MOV     R0,R13
000402AA: 1A10          DMOVB  R13,@010
16:          for(i=0;i<cnt;i++);
000402AC: C000          LDI:8   #00,R0
000402AE: 3FF0          ST      R0,@(R14,-4)
000402B0: 4050          LDUH   @(R14,10),R0
000402B2: 97A0          EXTSH  R0
```

In these lines clicking into the circle can set a breakpoint or right click into the circle, click on Break Point Set/Reset. The symbol then turns to

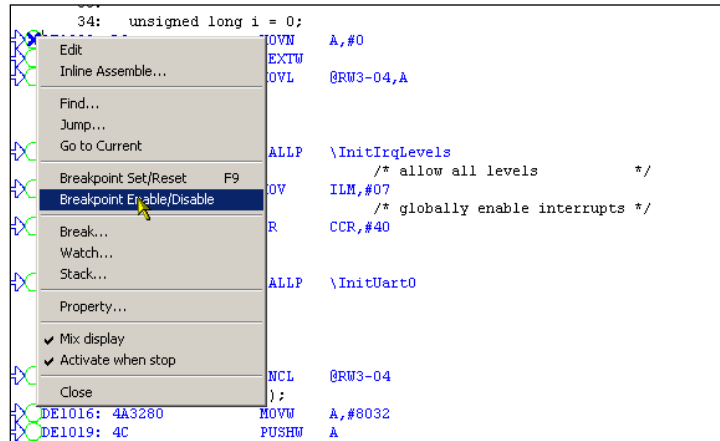
```
12: void wait (short int cnt)
000402A0: 1704          ST      R4,@-R15
000402A2: 1781          ST      RP,@-R15
000402A4: 0F02          ENTER  #008
13: {
14:          int i;
15:          PDR16 = 0xFF;
000402A6: CFF0          LDI:8   #FF,R0
000402A8: 8B0D          MOV     R0,R13
000402AA: 1A10          DMOVB  R13,@010
16:          for(i=0;i<cnt;i++);
000402AC: C000          LDI:8   #00,R0
000402AE: 3FF0          ST      R0,@(R14,-4)
000402B0: 4050          LDUH   @(R14,10),R0
000402B2: 97A0          EXTSH  R0
```

Depending on the selected break point type, a differently colored cross is shown in the circle. For the software break point (default), the cross is blue, while there is a red cross in the circle for the hardware break point

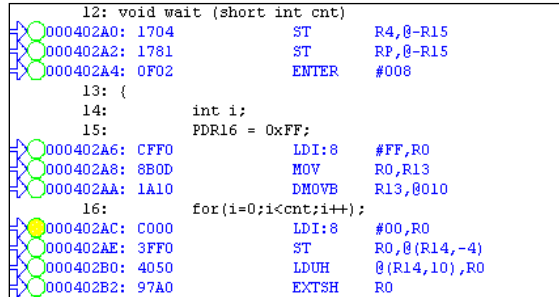
If you now start the execution, the CPU will halt on this break point. The actual line gets a yellow background color.

Clicking to this circle again or right clicking into the circle and clicking on Break Point Set/Reset, releases the break point.

Alternately, break point can be disabled by right clicking and on popup menu clicking Break Point enable/Disable.



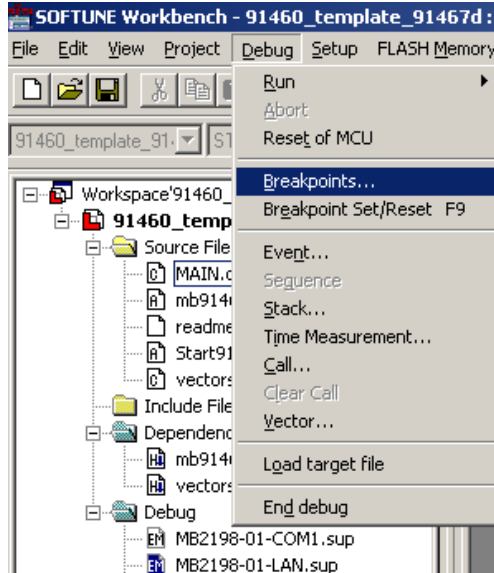
The symbol will change to



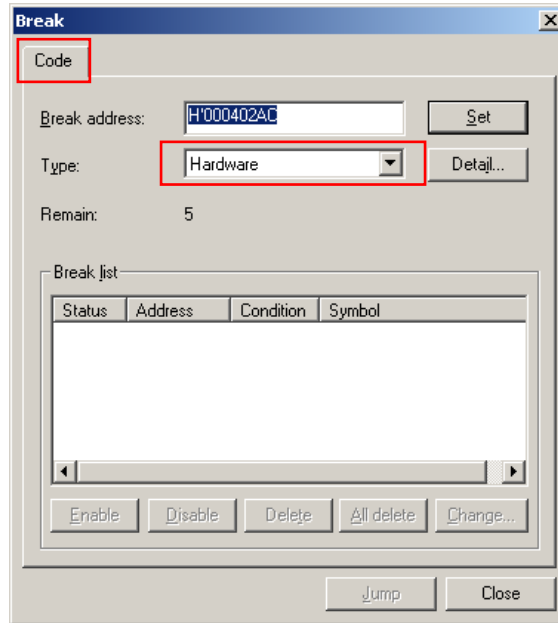
### 5.1.2 Setting code break point using Dialog box

A code break point can also be set using Dialog box

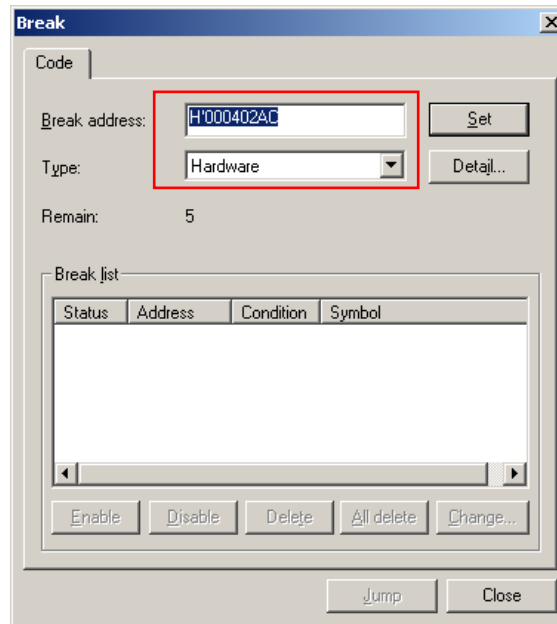
Open break point dialog box by clicking on *Breakpoints...* on Debug menu.



On code tab, select *software* or *Hardware* type breakpoint.



Type desired Break address



Click on set, to set software or Hardware breakpoint at required address

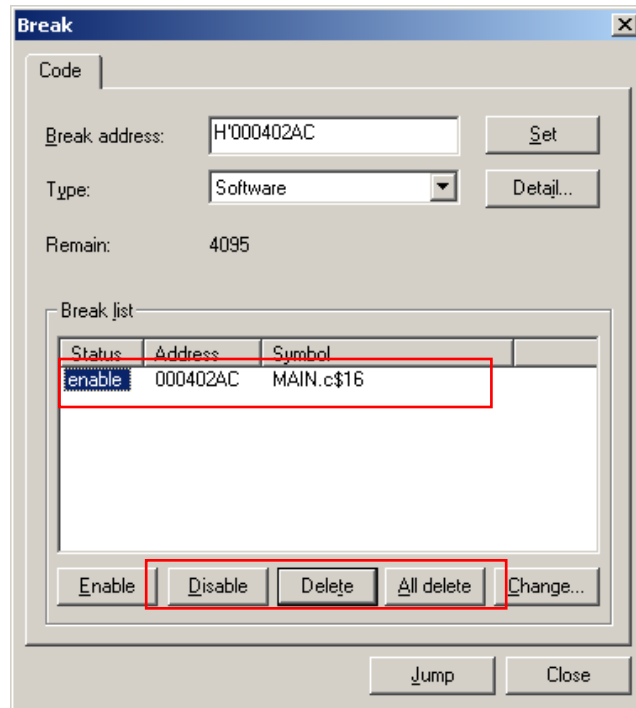
If you now start the execution, the CPU will halt on this break point. The actual line gets a yellow background color.

```

12: void wait (short int cnt)
000402A0: 1704          ST      R4,0-R15
000402A2: 1781          ST      RP,0-R15
000402A4: 0F02          ENTER   #008
13: {
14:     int i;
15:     PDR16 = 0xFF;
000402A6: CFF0          LDI:8   #FF,R0
000402A8: 8B0D          MOV     R0,R13
000402AA: 1A10          DMOVb  R13,@010
16:     for(i=0;i<cnt;i++);
000402AC: C000          LDI:8   #00,R0
000402AE: 3FF0          ST      R0,@(R14,-4)
000402B0: 4050          LDUH   @(R14,10),R0
000402B2: 97A0          EXTSH  R0
  
```



Breakpoint can be released by selecting breakpoint from *Break list*, and clicking on *Delete* button, alternately it can also be disabled by clicking *Disable* button

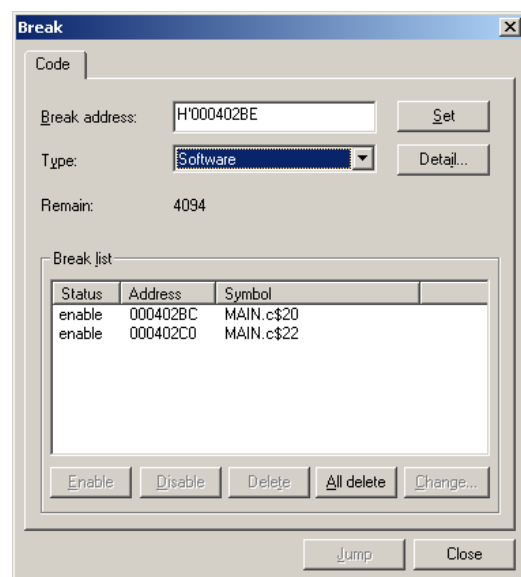
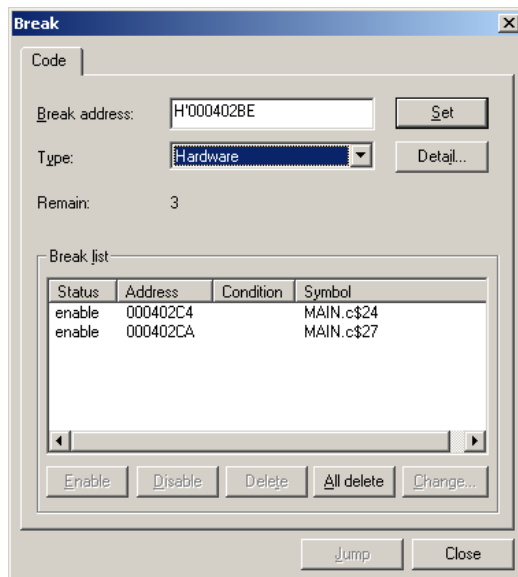


## 5.2 Position of Break point

When break points are set to some location and after that code is modified and build again, position of previously set break point will be decided as per following rule

- Break point will be set to the same source code line number
- If it is not possible to meet above condition, it will be set to the same address location

For example,



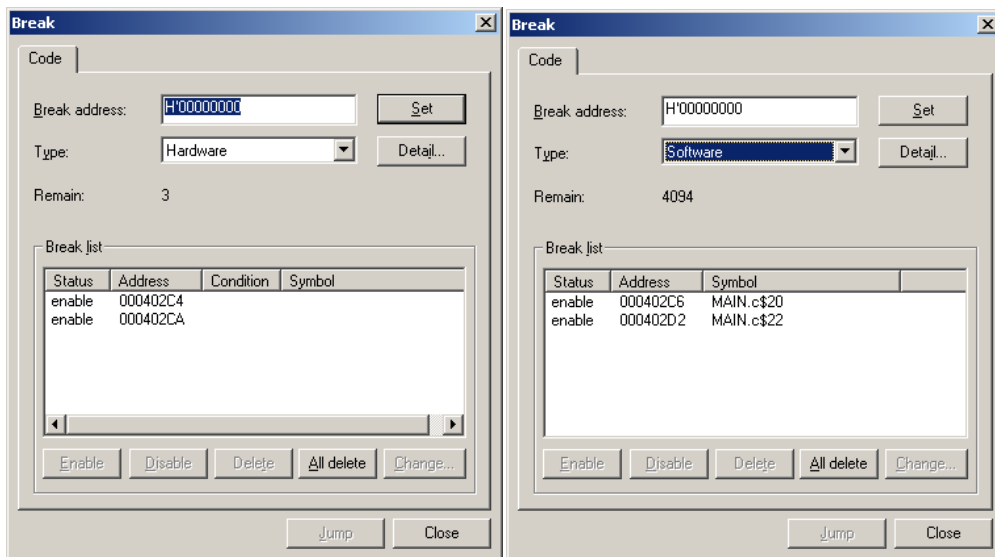
```

12: void wait (short int cnt)
13: {
14:     int i;
15:     PDR16 = 0xFF;
16:     for(i=0;i<cnt;i++)
17:     {
18:
19:         __asm(" NOP");
20:         __asm(" NOP");
21:         __asm(" NOP");
22:         __asm(" NOP");
23:         __asm(" NOP");
24:         __asm(" NOP");
25:         __asm(" NOP");
26:         __asm(" NOP");
27:         __asm(" NOP");
28:         __asm(" NOP");
29:
30:     }
31:
32:     PDR16 =0x00;
33:
34: }
35:

```

Let us change the code, replace couple of `__asm( "NOP") ;` instruction with some other instruction in `wait` function.

And let us assume that Breakpoint position changes as shown below



```

12: void wait (short int cnt)
000402A0: 1704          ST      R4,@-R15
000402A2: 1781          ST      RP,@-R15
000402A4: 0F02          ENTER   #008
13: {
14:     int i;
15:     PDR16 = 0xFF;
000402A6: CFF0          LDI:8   #FF,R0
000402A8: 8B0D          MOV     R0,R13
000402AA: 1A10          DMOVB  R13,@010
16:     for(i=0;i<cnt;i++)
000402AC: C000          LDI:8   #00,R0
000402AE: 3FF0          ST      R0,@(R14,-4)
000402B0: 4050          LDUH   @(R14,10),R0
000402B2: 97A0          EXTSH  R0
000402B4: 2FF1          LD     @(R14,-4),R1
000402B6: AA01          CMP    R0,R1
000402B8: EB13          BGE    000402E0
17:     {
18:         i = i+1;
000402BA: 2FF0          LD     @(R14,-4),R0
000402BC: A410          ADD    #1,R0
000402BE: 3FF0          ST     R0,@(R14,-4)
19:         i = i+2;
000402C0: 2FF0          LD     @(R14,-4),R0
000402C2: A420          ADD    #2,R0
000402C4: 3FF0          ST     R0,@(R14,-4)
20:         i = i+3;
000402C6: 2FF0          LD     @(R14,-4),R0
000402C8: A430          ADD    #3,R0
000402CA: 3FF0          ST     R0,@(R14,-4)
21:         i = i+4;
000402CC: 2FF0          LD     @(R14,-4),R0
000402CE: A440          ADD    #4,R0
000402D0: 3FF0          ST     R0,@(R14,-4)
22:         i = i+5;
000402D2: 2FF0          LD     @(R14,-4),R0
000402D4: A450          ADD    #5,R0
000402D6: 3FF0          ST     R0,@(R14,-4)
000402D8: 2FF0          LD     @(R14,-4),R0
000402DA: A410          ADD    #1,R0
000402DC: 3FF0          ST     R0,@(R14,-4)
000402DE: E0E8          BRA    000402B0
23:     }
24:     }
25:     PDR16 =0x00;
000402E0: C000          LDI:8   #00,R0
000402E2: 8B0D          MOV     R0,R13
000402E4: 1A10          DMOVB  R13,@010
26:
27: }

```

For Software breakpoint 1, this was at source code line no. 20, since after build there is valid instruction at that line; breakpoint is maintained at same source code line main.c\$20

For Software breakpoint 2, this was at source code line no. 22, since after build there is valid instruction at that line; breakpoint is maintained at same source code line main.c\$22

For Hardware breakpoint 3, this was at source code line no. 24, since after build there is no valid instruction at that line; breakpoint is maintained at same memory location H' 000402C4

For Hardware breakpoint 4, this was at source code line no. 27, since after build there is no valid instruction at that line; breakpoint is maintained at same memory location H' 000402CA

# 6. Events



## 6.1 How to Set Code and Data Events

Two types of Event can be set. Those are Code event and Data event.

### Code Event:

Two code events can be set.

Code event can be set with *pass count* from 1 to 65535 and *address mask*.

Pass count: It is the count of times PC needs to pass the set Address before program execution stops at the *address*.

Address Mask: A pattern of characters, 32 bits long, used to select some of the bits from the *address*. For each bit set to one in *Address mask* field, corresponding bit from Address is compared for exact match with the corresponding bit hold by PC and for each bit set to zero, corresponding bit from Address is ignored. Detail explanation with example is given in coming chapters.

### Data Event:

When Data event is set, program execution stops when the data at the *address* is accessed.

As explained in code event it is possible to set data event point with *Address mask*

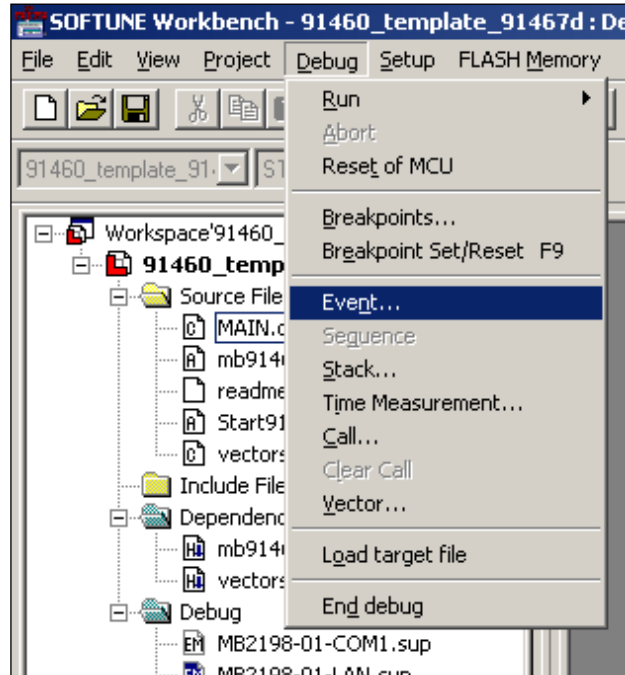
It is also possible to specify the Data content with *Data mask*, which is when read from or written to the Address, program execution will halt.

Further data can be specified with *size* option i.e. byte, half word and word.

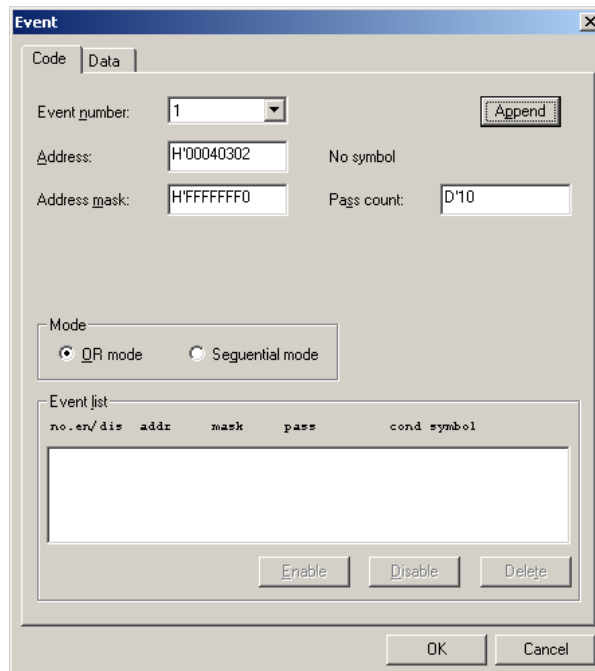
Two data event can be set.

### 6.1.1 Code Event

As shown below, *Event* dialog box is opened by clicking on *Event...* option on *Debug* pull down menu.

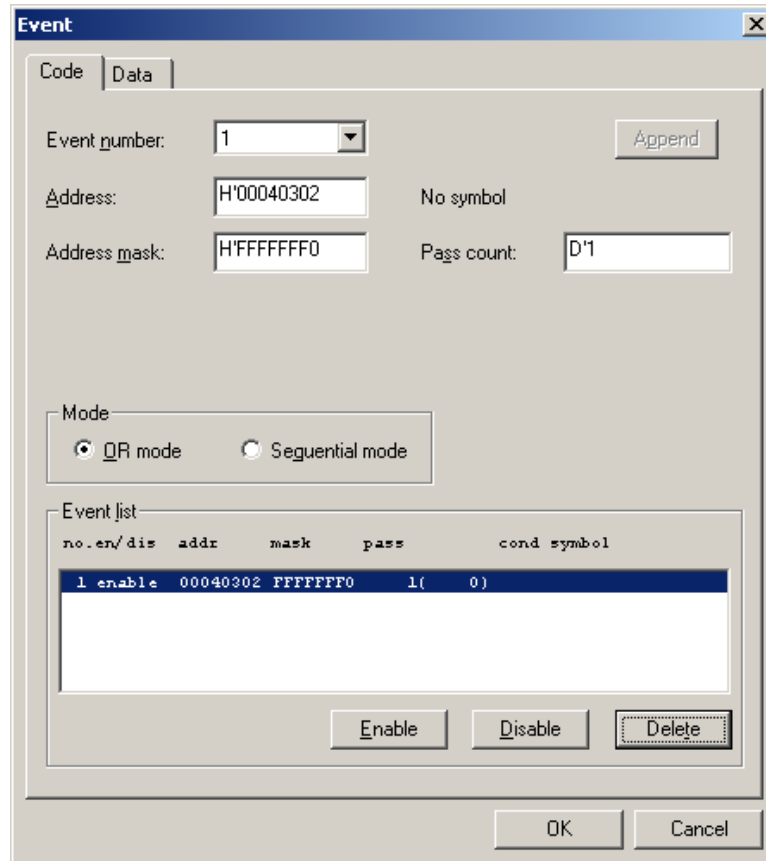


In the below dialog box, *Event Address* - Specifies the address at which the event occurrence condition is to be set. *Address Mask* specifies address mask. Under *Mode*, two possible modes can be selected. In *OR Mode*, events are triggered when the event 1 condition or the event 2 condition is established. In *Sequential mode* Events are triggered when the event 1 condition and event 2 condition are established in sequence.



When *address mask* is selected, for each bit set in *address mask* field, corresponding bit from *Address* is compared for the exact match with the corresponding bit of address hold by Program Counter i.e. Program will stop executing due to event break whenever following condition is met...

$$(\text{Address hold by PC}) \& (\text{Address Mask}) = (\text{address}) \& (\text{Address Mask})$$



Event

Code | Data

Event number: 1 [Append]

Address: H'00040302 No symbol

Address mask: H'FFFFFFF0 Pass count: D'1

Mode:  
 QR mode  Sequential mode

no.	en/dis	addr	mask	pass	cond	symbol
1	enable	00040302	FFFFFFF0	1	( 0)	

[Enable] [Disable] [Delete]

[OK] [Cancel]

For example, if we set Code Breakpoint at address H' 00040302 and address Mask H'FFFFFFF0 and when program is executed it will stop at all the address (where there is valid instruction) from H' 00040300 to H' 0004030F

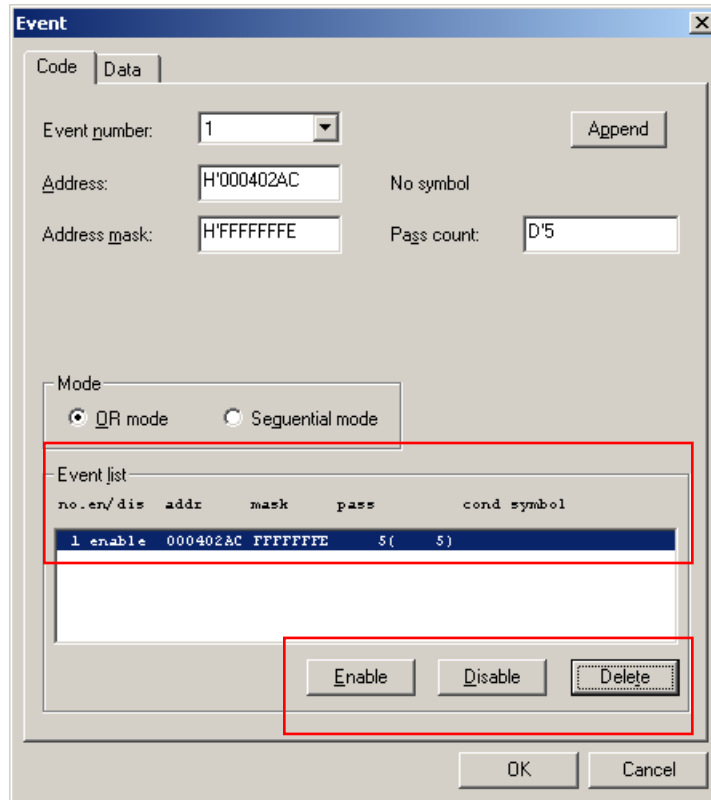
Similarly when we select *Pass count* (1 to 255), program will halt executing at *Address* when program counter executes instruction at the address selected in the *Address* field, as many numbers of times as that is selected by *Pass count*.

For example, if we set code break point at Address H'000402AC and select Pass count equal to D'5. When program is run, it halts due to hardware code break point when value of variable 'i' equal to D'5

```

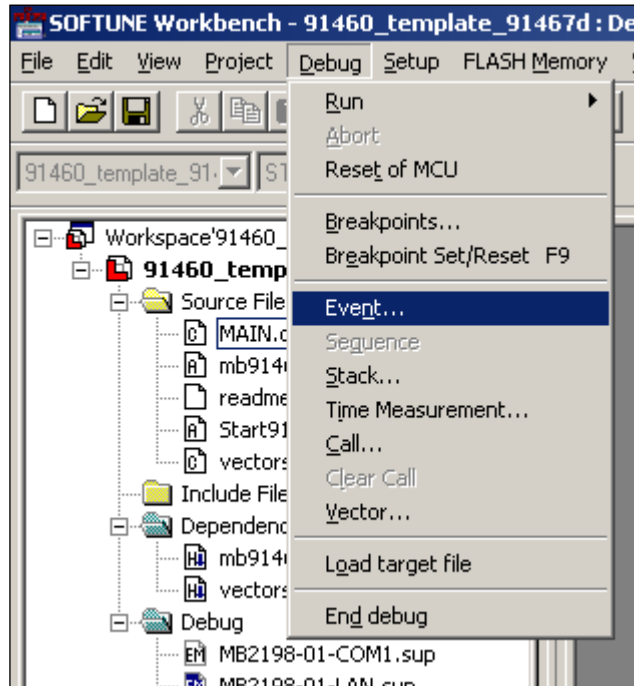
12: void wait (short int cnt)
000402A0: 1704      ST      R4,@-R15
000402A2: 1781      ST      RP,@-R15
000402A4: 0F02      ENTER  #008
13: {
14:     int i;
15:     PDR16 = 0xFF;
000402A6: CFF0      LDI:8   #FF,R0
000402A8: 8B0D      MOV     R0,R13
000402AA: 1A10      DMOVB  R13,@010
16:     for(i=0;i<cnt;i++);
000402AC: C000      LDI:8   #00,R0
000402AE: 3FF0      ST      R0,@(R14,-4)
000402B0: 4050      LDUH   @(R14,10),R0
000402B2: 97A0      EXTSH  R0
000402B4: 2FF1      LD     @(R14,-4),R1
000402B6: AA01      CMP    R0,R1
000402B8: EB04      BGE    000402C2
000402BA: 2FF0      LD     @(R14,-4),R0
000402BC: A410      ADD    #1,R0
000402BE: 3FF0      ST      R0,@(R14,-4)
000402C0: EOF7      BRA    000402B0
  
```

Event can be released by selecting it from Event list, and clicking on Delete button, alternately it can also be disabled by clicking Disable button

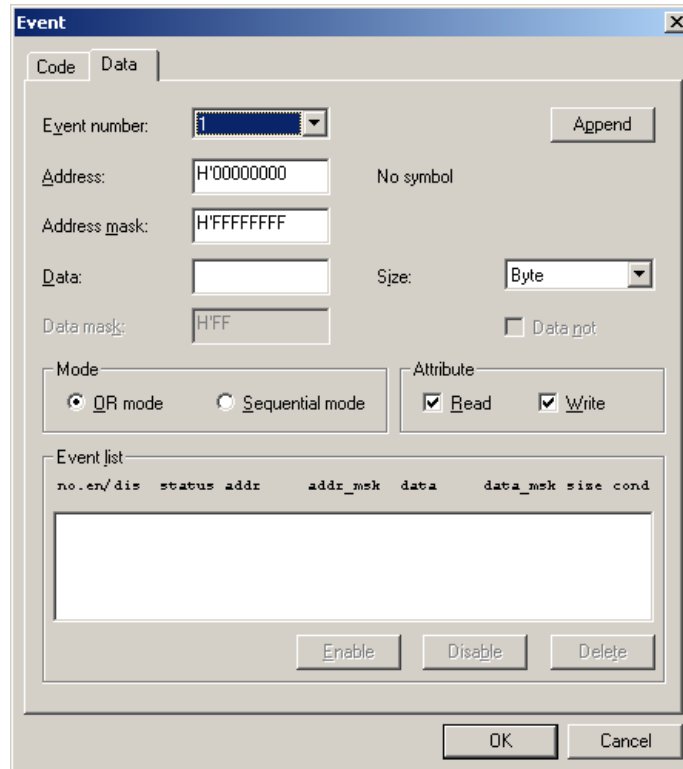


### 6.1.2 Data Event

As shown below, *Break* dialog box is opened by clicking on *breakpoints...* option on *Debug* pull down menu.



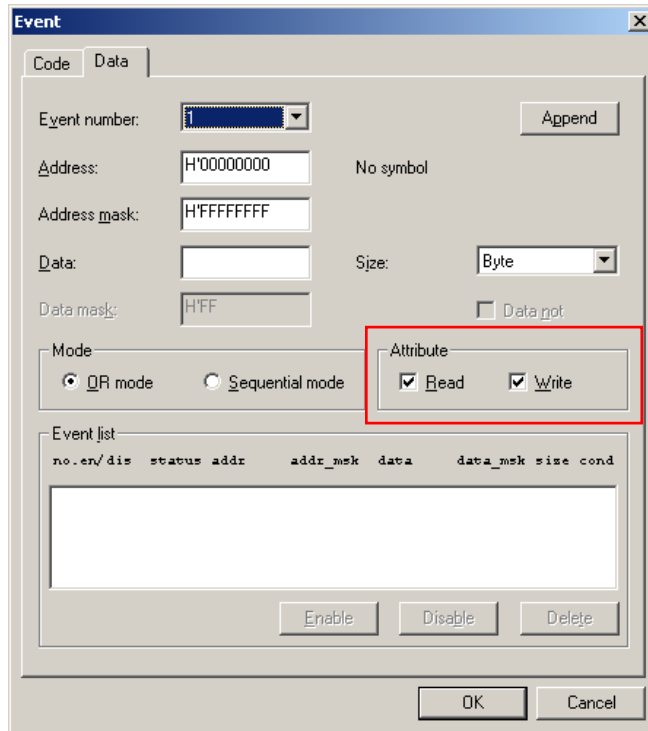
Click on *Data* tab,



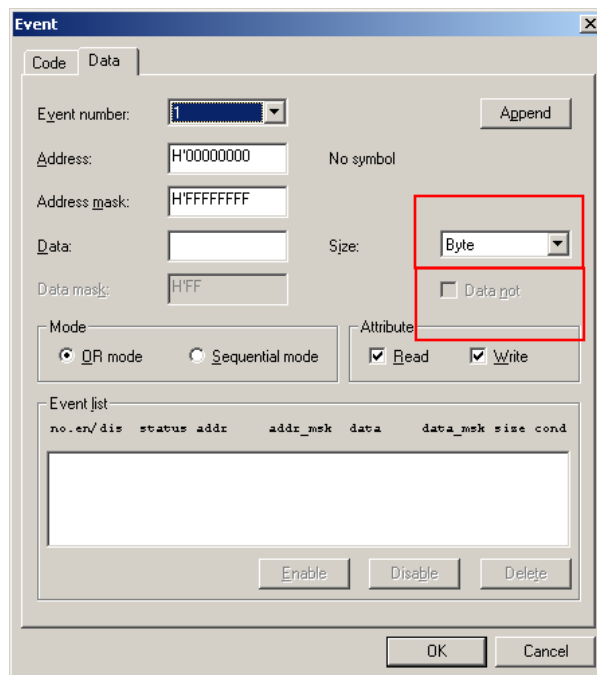


Address mask and Pass count feature work same as that of code event. *Data* specifies the data to be set as the event occurrence condition. *Data mask* specifies data mask. *Data not* specifies the condition when the data values do not match. *Size* specifies a data access size. *Attribute* specifies a data access attribute.

In *Event* dialog box, under *attribute* option it can be specified if *read* or *write* access at *Address* should cause a break.



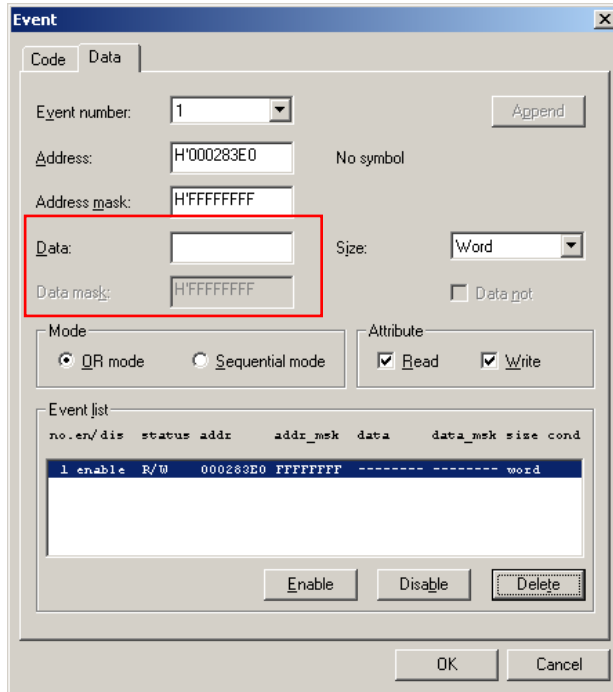
Under *size* option it is possible to select either *byte*, *halfword* or *word* type data.



Further, comparison condition can be selected among *Data equal to* or *Data not*.

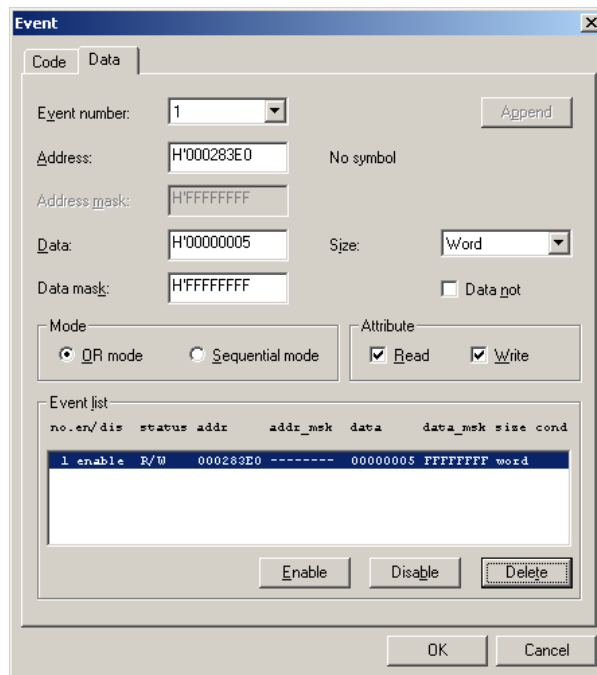
Note, that the Data and Data mask text boxes are only available if either *Data agreement* or *Data not* is selected.

When Data field is left empty, for all data access at Address for read or write access as selected by attribute, program execution will halt.



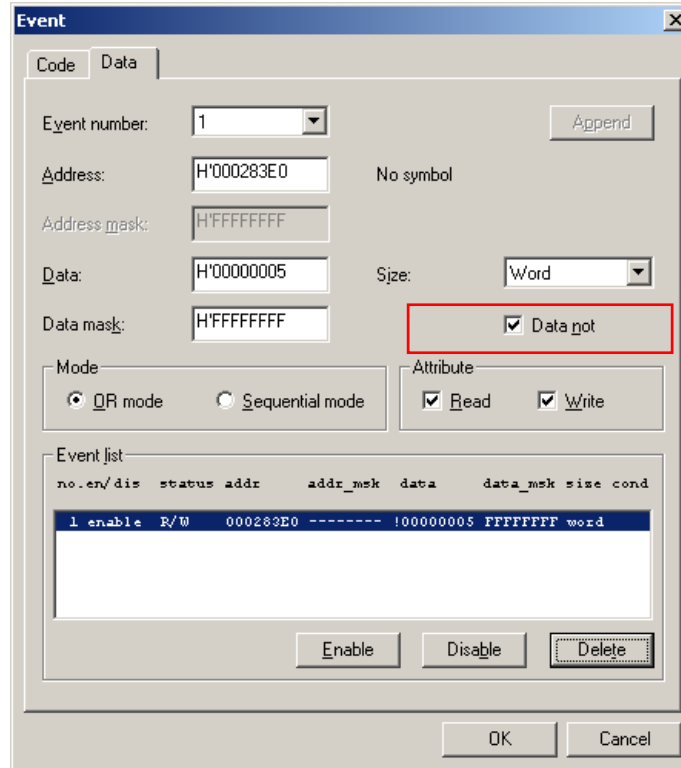
For example, if we set breakpoint as shown in above figure, program execution will halt every time when there is a read or write access to location H'000283E0

When *Data not* check box is unselected, for all data access at Address, the content *read* from or *written* to, is equal to that specified in Data field, program execution will halt.



For example, if we set breakpoint as shown in above figure, program execution will halt every time when data H' 05 is read from or written to location H'000283E0.

Similarly if we select *Data not* check box, for all data access at *Address*, the content *read* from or *written* to, is not equal to that specified in *Data* field, program execution will halt.

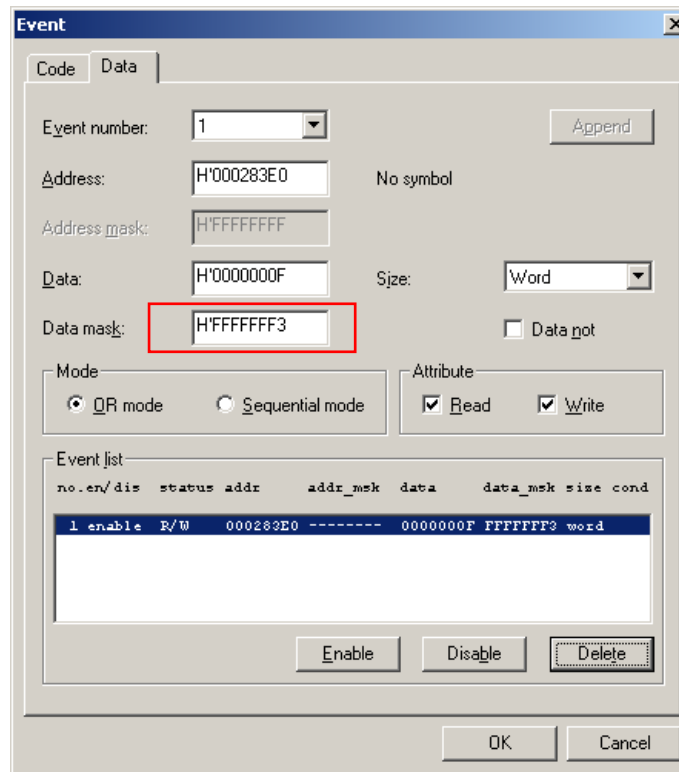


For example, if we set breakpoint as shown in above figure, program execution will halt every time when data other than H' 05 is read from or written to location H'000283E0.

When *Data mask* is selected, for each bit set in *Data mask* field, that bit from *Data* field is compared for the exact match with the corresponding bit of Data which is to be read or written at the break address by the MCU i.e. Program will stop executing due to break point whenever following condition is met...

$$(\text{Data read by MCU at Break Address Selected}) \& (\text{Data Mask}) = (\text{Data}) \& (\text{Data Mask})$$

For example, if we set Data Breakpoint at address H' 000283E0, which is the address of a int type variable and if the attribute is set to read, comparison condition is set to Data agreement, data is set to H'000000F and Data Mask is set to H'FFFFFF3.



When program is executed it will stop when data, either H'00000C or H'00000D or H'00000E or H'00000F is read from break address H' 000283E0.

Now let us consider that we want in a following code, we want program to stop executing when `count = H'05` and `i = H'FF`. We can do this using sequential mode, while setting an event

```

void wait (short int cnt)
{
    int i;
    PDR16 = 0xFF;
    for(i=0;i<cnt;i++);
    PDR16 = 0x00;
}

void main(void)
{
    int count = 0;

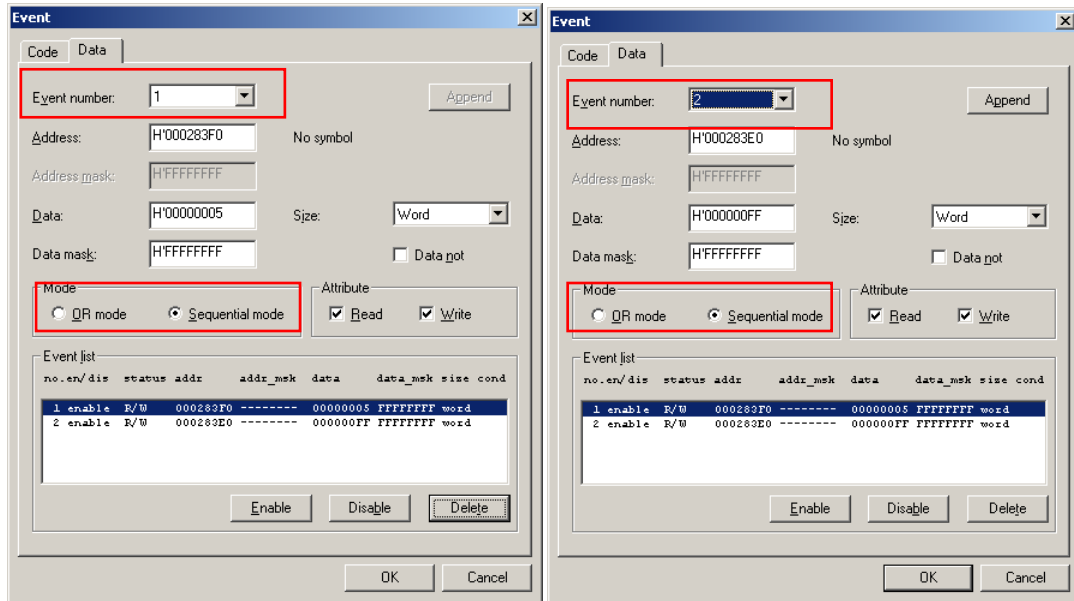
    __EI();                /* enable interrupts */
    __set_il(31);          /* allow all levels */
    InitIrqLevels();       /* init interrupts */

    PORTEN = 0x3;         /* enable I/O Ports */
                          /* This feature is not supported by MB91U460A */
                          /* For all other devices the I/O Ports must be enabled*/

    PFR16 = 0x00;
    DDR16 = 0xFF;

    while(1)              /* endless loop */
    {
        HWWD_CL = 0;
        count++;
        __asm(" NOP");
        wait (5000);
        __asm(" NOP");
    }
}
    
```

For this we need to set event 1 and 2 as described below, assuming *count* is located at H'000283F0 and *i* is located at H'000283E0



# 7. Trace



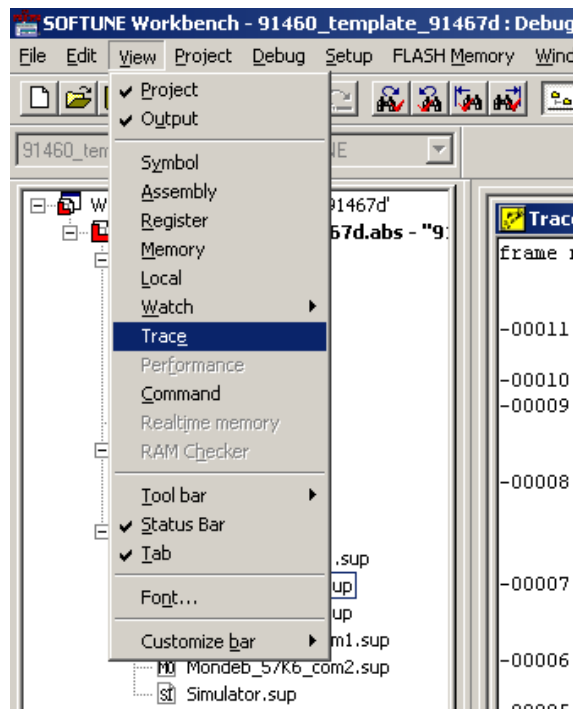
## How To Use The Trace Buffer

Trace function displays addresses and instructions executed so far. While execution of a program, the address, data and status information can be sampled and stored in the trace buffer. This function is called real-time trace.

In-depth analysis of a program execution history can be performed using the data recorded by real-time trace.

## 7.1 Trace Window

Trace window can be opened by clicking *Trace*, on *View* pull down menu.



```

Trace
-----
frame no.  address  mnemonic (-65535 .. 00000)
:          :          :          :
: 000402BC ADD      #1,R0
: 000402BE ST       R0,@(R14,-4)
-00011 : write 00000748 at 000283E0
:          :          :          :
: 000402C0 BRA      000402B0
-00010 : 000402B0 LDUH    @(R14,10),R0
-00009 : read   1388 at 000283EE
:          :          :          :
: 000402B2 EXTSH   R0
: 000402B4 LD      @(R14,-4),R1
-00008 : read   00000748 at 000283E0
:          :          :          :
: 000402B6 CMP     R0,R1
: 000402B8 BGE     000402C2
: 000402BA LD      @(R14,-4),R0
-00007 : read   00000748 at 000283E0
:          :          :          :
: 000402BC ADD     #1,R0
: 000402BE ST      R0,@(R14,-4)
-00006 : write 00000749 at 000283E0
:          :          :          :
: 000402C0 BRA     000402B0
-00005 : 000402B0 LDUH    @(R14,10),R0
-00004 : read   1388 at 000283EE
  
```

In a Trace result display, the first column shows the frame number.

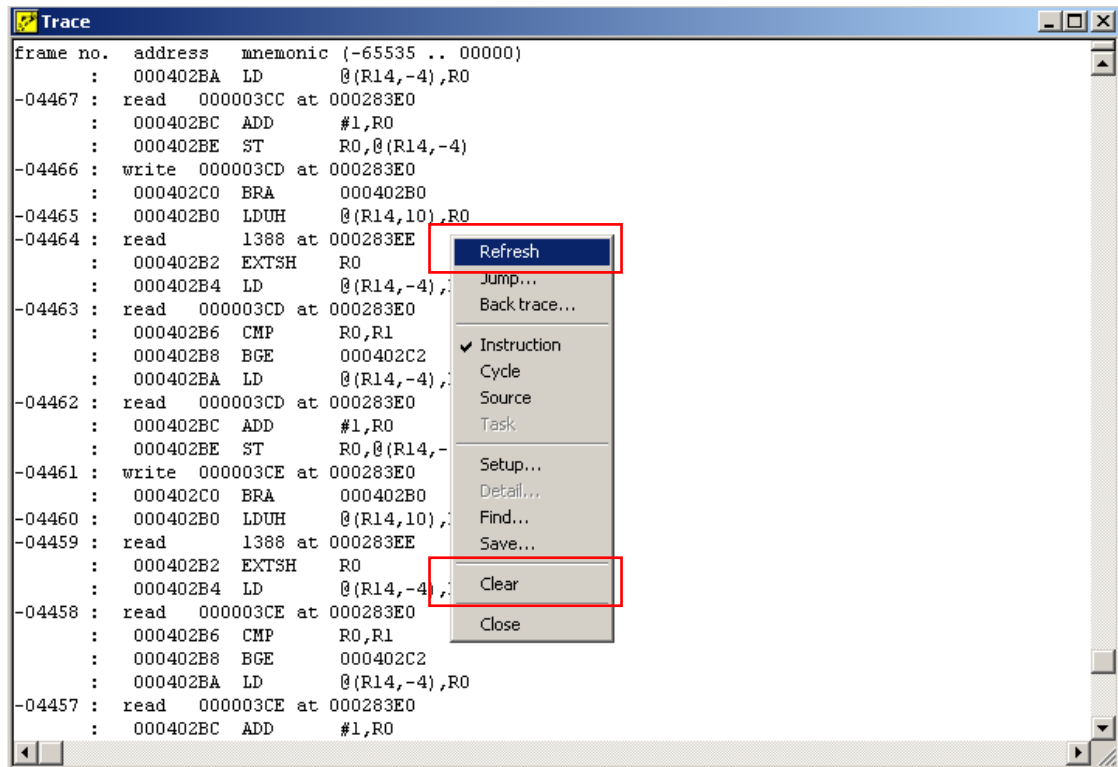
The second column shows the internal bus address.

The third column shows either the disassembled machine code (mnemonic) or a data transfer (inclusive data value).

Note that the last executed frame has the number 0 and all previous frames negative numbers.

Every time, after program run, to refresh a trace window right click on Trace Window, on a popup menu click on *Refresh*.

To clear previous data, right click on Trace Window, on a popup menu click on *Clear*.



```

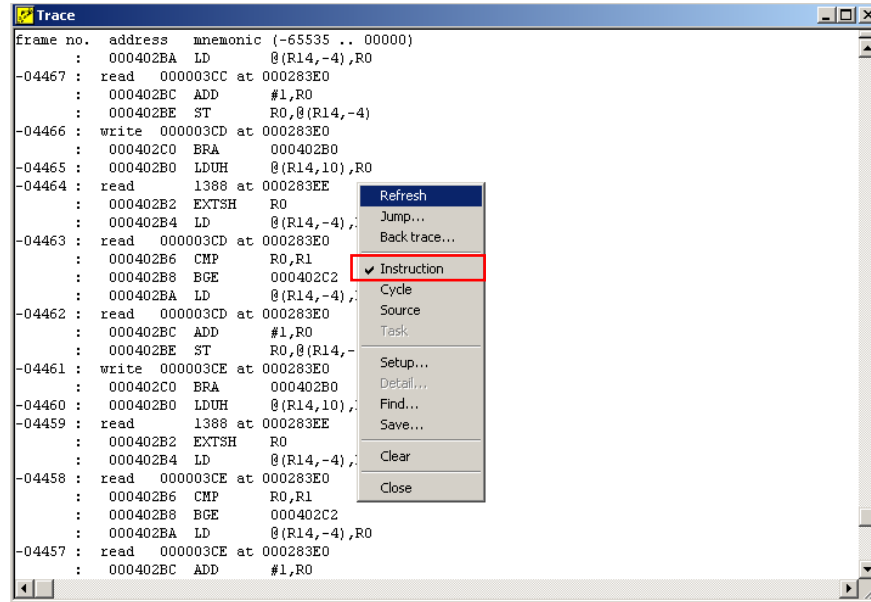
Trace
-----
frame no.  address  mnemonic (-65535 .. 00000)
:          :          :          :
: 000402BA LD      @(R14,-4),R0
-04467 : read   000003CC at 000283E0
:          :          :          :
: 000402BC ADD     #1,R0
: 000402BE ST      R0,@(R14,-4)
-04466 : write 000003CD at 000283E0
:          :          :          :
: 000402C0 BRA     000402B0
-04465 : 000402B0 LDUH    @(R14,10),R0
-04464 : read   1388 at 000283EE
:          :          :          :
: 000402B2 EXTSH   R0
: 000402B4 LD      @(R14,-4),R1
-04463 : read   000003CD at 000283E0
:          :          :          :
: 000402B6 CMP     R0,R1
: 000402B8 BGE     000402C2
: 000402BA LD      @(R14,-4),R0
-04462 : read   000003CD at 000283E0
:          :          :          :
: 000402BC ADD     #1,R0
: 000402BE ST      R0,@(R14,-4)
-04461 : write 000003CE at 000283E0
:          :          :          :
: 000402C0 BRA     000402B0
-04460 : 000402B0 LDUH    @(R14,10),R0
-04459 : read   1388 at 000283EE
:          :          :          :
: 000402B2 EXTSH   R0
: 000402B4 LD      @(R14,-4),R1
-04458 : read   000003CE at 000283E0
:          :          :          :
: 000402B6 CMP     R0,R1
: 000402B8 BGE     000402C2
: 000402BA LD      @(R14,-4),R0
-04457 : read   000003CE at 000283E0
:          :          :          :
: 000402BC ADD     #1,R0
  
```

## 7.2 Trace View

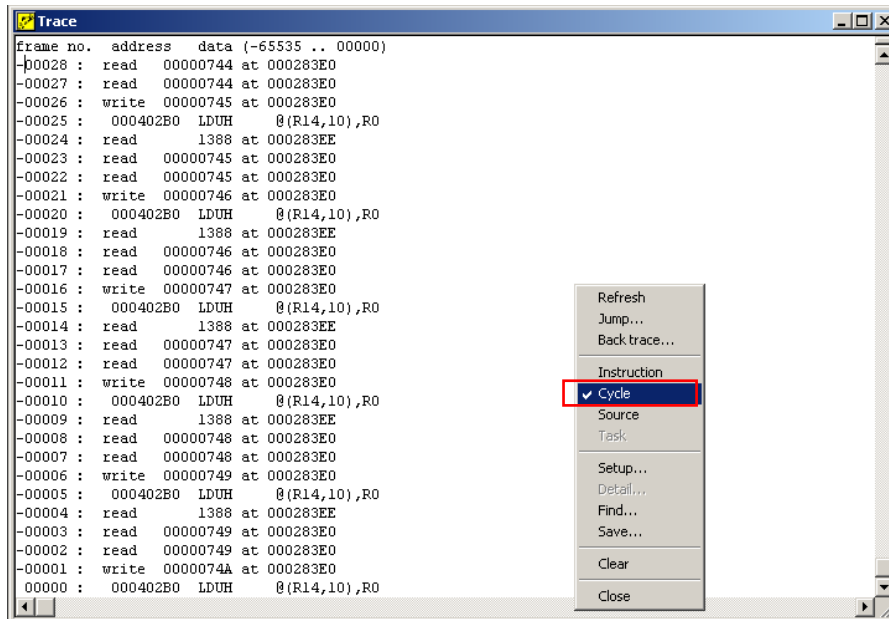
Trace view can be changed among Instruction view, raw data view or source view.

For Instruction view, right click on Trace window, on a popup menu click on *Instruction*.

Instruction view show instruction cycles using assembler mnemonics



For raw data view, right click on Trace window, on a popup menu click on *Cycle*.

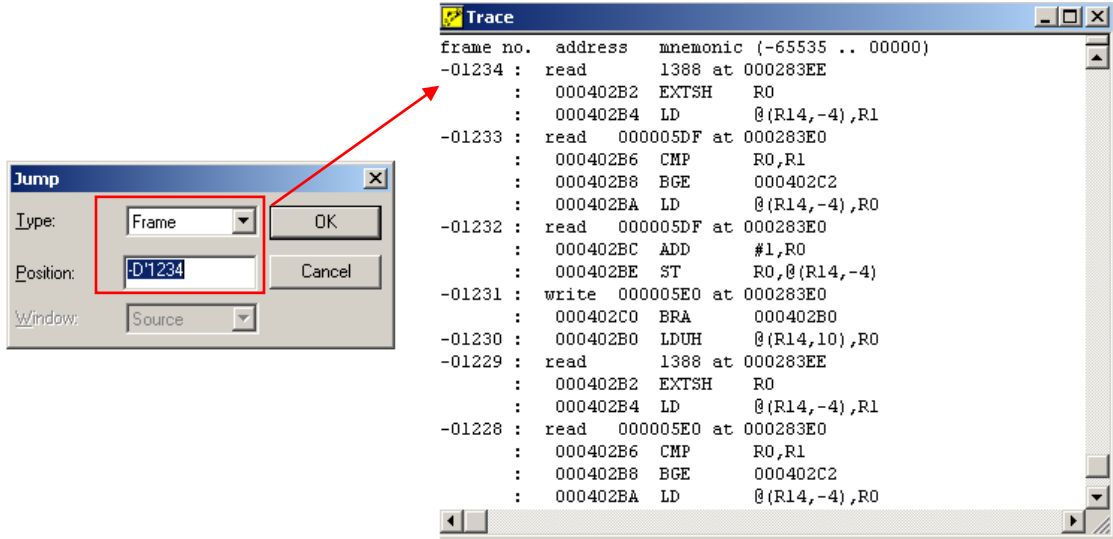


For Source view, right click on Trace window, on a popup menu click on *Source*. Source view shows program source code (e.g. C - language)

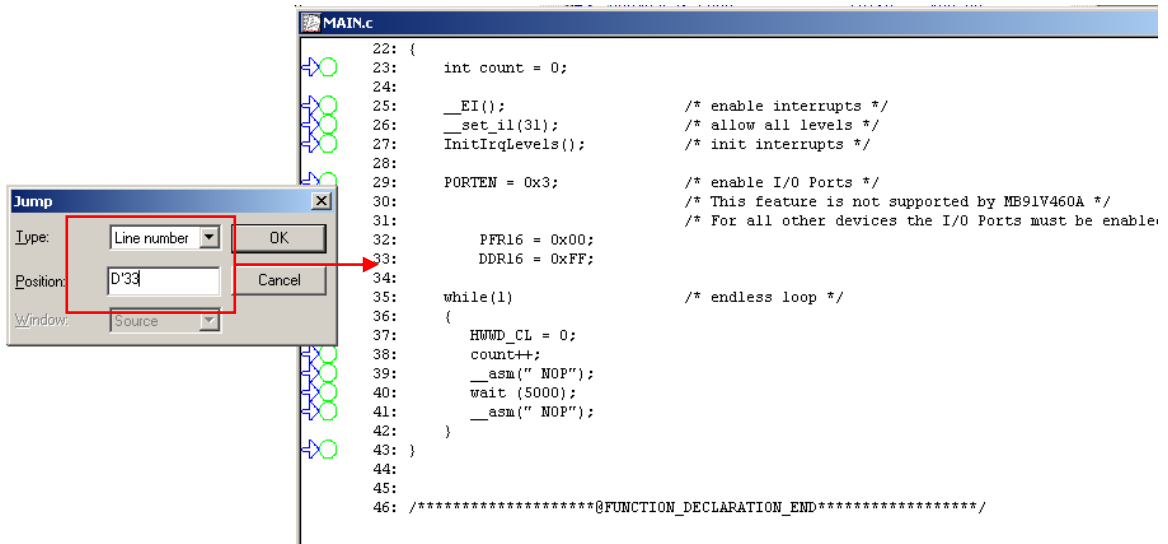


### 7.3 Trace Jump

To move cursor to particular frame in trace window, right click on *trace* window, click on *Jump...*, on *Jump* dialog box, select *frame* under *Type* option and desired frame number under *Position* option and click *OK*



To view source code at particular line number, right click on trace window, click on *Jump...*, on *Jump* dialog box, select *Line number* under *Type* option and source line number under *Position* option, click *OK*, in editor window one can view source code at selected line number



To view assembly instruction at particular memory location, right click on trace window, click on *Jump...*, on *Jump* dialog box select *Address* under *Type* option and memory location under *Position* option and then select either *Source* or *Assembly* under *Window* option.

**Jump Dialog Box (Top):**

- Type: Address
- Position: H'000402EE
- Window: Source

**Assembly Window (Right):**

```

MAIN.c
27:   InitIrqLevels();           /* init interrupts */
000402DC: 9F8C00040000   LDI:32  #00040000,R12
000402E2: 971C           CALL   @R12
28:
29:   PORTEN = 0x3;              /* enable I/O Ports */
000402E4: C030           LDI:8   #03,R0
000402E6: 9F8C00000498   LDI:32  #00000498,R12
000402EC: 16C0           STB    R0,@R12
30:
31:   /* This feature is no
32:   /* For all other devi
000402EE: C000           LDI:8   #00,R0
000402F0: 9F8C00000D90   LDI:32  #00000D90,R12
000402F6: 16C0           STB    R0,@R12
  
```

**Jump Dialog Box (Bottom):**

- Type: Address
- Position: H'000402EE
- Window: Assembly

**Assembly Window (Right):**

```

Assembly
000402EE: C000           LDI:8   #00,R0
000402F0: 9F8C00000D90   LDI:32  #00000D90,R12
000402F6: 16C0           STB    R0,@R12
000402F8: CFF0           LDI:8   #FF,R0
000402FA: 9F8C00000D50   LDI:32  #00000D50,R12
00040300: 16C0           STB    R0,@R12
00040302: 9F80000004C7   LDI:32  #000004C7,R0
00040308: 8070           BANDL  #7,@R0
0004030A: 2FF0           LD     @(R14,-4),R0
0004030C: A410           ADD   #1,R0
0004030E: 3FF0           ST    R0,@(R14,-4)
00040310: 9FA0           NOP
00040312: 9B041388       LDI:20  #01388,R4
  
```

To view data at particular memory location, right click on trace window, click on *Jump...*, on *Jump* dialog box select *Address* under *Type* option and memory location under *Position* option and then select *memory* under *window* option.

**Jump Dialog Box (Left):**

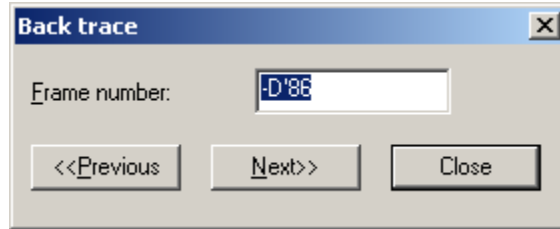
- Type: Address
- Position: H'000402B2
- Window: Memory

**Memory Window (Right):**

Address	+0	+2	+4	+6	+8	+A	Ascii
000402B2	97A0	2FF1	AA01	EB04	2FF0	A410	.../...../...
000402BE	3FF0	E0F7	C000	8B0D	1A10	9F90	?.....
000402CA	0781	A301	9720	1781	0F02	C000	.....
000402D6	3FF0	9310	871F	9F8C	0004	0000	?.....
000402E2	971C	C030	9F8C	0000	0498	16C0	...0.....
000402EE	C000	9F8C	0000	0D90	16C0	CFF0	.....
000402FA	9F8C	0000	0D50	16C0	9F80	0000	....P.....
00040306	04C7	8070	2FF0	A410	3FF0	9FA0	...p/...?...
00040312	9B04	1388	D7C4	9FA0	E0F3	9F90	.....
0004031E	0781	9720	A410	3FF0	9F80	0000	...?.....
0004032A	04C7	8070	9FA0	9B04	1388	D7B5	...p.....
00040336	9FA0	E0F3	9F90	0781	9720	4480	..... D.
00040342	5109	001A	29E8	1CE1	0700	5284	Q...}.....R.
0004034E	4A04	0082	2800	0075	0C3B	0008	J...{..u... .
0004035A	9141	42D8	0601	0083	2430	2A41	.AB.....\$0*A

## 7.4 Back Trace

When selected *Source view*, one can use *Back trace...* functionality. Right click on editor window, click on *Back trace...*, *Back trace* dialog box will appear



With this functionality, one can traverse back and forth in source window in a same sequence, in which source code is executed. Source code at particular frame number will be highlighted by pink background while traversing.

For example, in figure below, trace window shows address and instruction at frame number -D'2 and -D'4 executed in sequence. In a *back trace* dialog box, if we select *frame number* -D'4, corresponding line at D'1502 in source code will be highlighted, if we click next, source code line number D'1530 will be highlighted which corresponds to *frame number* -D'2.

**Trace Window**

```

-00022 : Start.asm@1502      BBC CKMR:6,no_Pll_yet      ; check PCM and wait for
-00020 : Start.asm@1502      BBC CKMR:6,no_Pll_yet      ; PLL to stabilize
-00018 : Start.asm@1502      BBC CKMR:6,no_Pll_yet      ; PLL to stabilize
-00016 : Start.asm@1502      BBC CKMR:6,no_Pll_yet      ; PLL to stabilize
-00014 : Start.asm@1502      BBC CKMR:6,no_Pll_yet      ; PLL to stabilize
-00012 : Start.asm@1502      BBC CKMR:6,no_Pll_yet      ; check PCM and wait for
-00010 : Start.asm@1502      BBC CKMR:6,no_Pll_yet      ; check PCM and wait for
-00008 : Start.asm@1502      BBC CKMR:6,no_Pll_yet      ; check PCM and wait for
-00006 : Start.asm@1502      BBC CKMR:6,no_Pll_yet      ; check PCM and wait for
-00004 : Start.asm@1502      BBC CKMR:6,no_Pll_yet      ; check PCM and wait for
-00002 : Start.asm@1530      CALLP main                  ; Start main function
  
```

**Source Code**

```

1500: (CLKDIV_SPEED == CPU_36MHZ_CLKFZ_14MHZ)
1501: no PLL yet;
1502: BBC CKMR:6,no_Pll_yet      ; check PCM and wait for
1503:                               ; PLL to stabilize
1504: #endif ; wait for PLL
1505:
1506: ;=====
1507: ; 6.12 Initialise Low-Level Library Int
1508: ;=====
1509: ;
1510: ; Call lib init function and reload stack
1511: ;=====
1512: #if CLIBINIT == ON
1513: # if MEMMODEL == SMALL || MEMMODEL == COMPACT
  
```

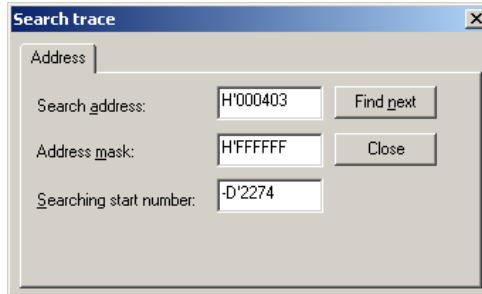
**Source Code**

```

1526: ;=====
1527: #if MEMMODEL == SMALL || MEMMODEL == COMPACT
1528: CALL _main                  ; Start main function
1529: #else
1530: CALLP _main                 ; Start main function
1531:                               ; ignore remaining word on stack,
1532:                               ; if main was completed by RET
1533: #endif
1534: ;=====
1535: ; 6.14 Shut down library
1536: ;=====
1537: #if CLIBINIT == ON
1538: # if MEMMODEL == SMALL || MEMMODEL == COMPACT
1539: CALL _exit                  ; Start main function
1540: # else
1541: CALLP _exit                 ; ignore remaining word on stack,
1542:                               ; if main was completed by RET
  
```

## 7.5 Search Trace

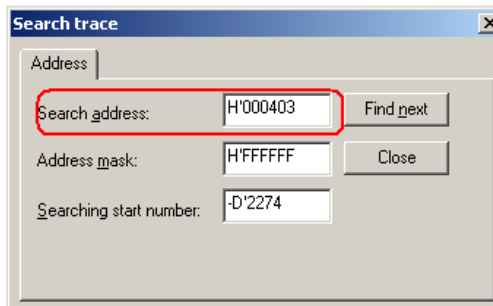
To find particular memory location in trace window, one can use *find* feature. Right click on Trace window, on a popup menu click on *Find*. *Search trace* dialog box will appear.



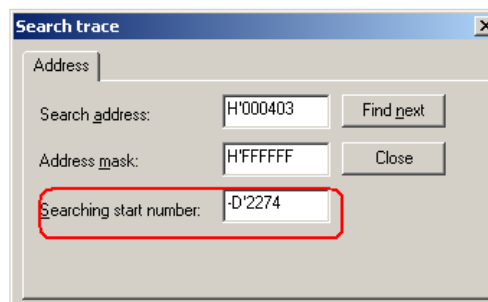
The line having information about searched memory location is highlighted with black background.

-01812 :	write	55AA	at 00253A	*	1
-01811 :	write	55AA	at 00253C	*	1
-01810 :	write	55AA	at 00253E	*	1
-01809 :	read	B0	at 000403	*	9
-01808 :	F800A2	BBC	0403:6,STARTUP\nno_PLL_yet*	*	1
-01807 :	read	B0	at 000403	*	9
-01806 :	F800A2	BBC	0403:6,STARTUP\nno_PLL_yet*	*	1
-01805 :	read	B0	at 000403	*	10
-01804 :	F800A2	BBC	0403:6,STARTUP\nno_PLL_yet*	*	1
-01803 :	read	B0	at 000403	*	9
-01802 :	F800A2	BBC	0403:6,STARTUP\nno_PLL_yet*	*	1
-01801 :	read	B0	at 000403	*	10
-01800 :	F800A2	BBC	0403:6,STARTUP\nno_PLL_yet*	*	1
-01799 :	read	B0	at 000403	*	9
-01798 :	F800A2	BBC	0403:6,STARTUP\nno_PLL_yet*	*	1
-01797 :	read	B0	at 000403	*	10

The required memory location is entered in *Search address*,



The frame number from where to start the trace search is entered in *Searching start number*

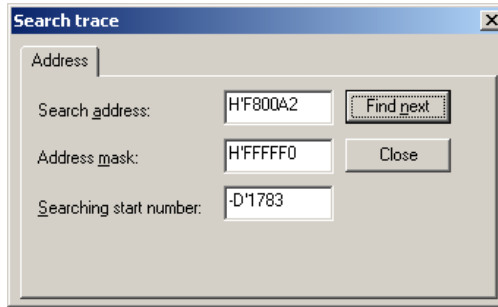


When *Address mask* is selected, for each bit set in *Address mask* field, that bit from *Search address* field is compared for the exact match with the corresponding bit of the memory location read by the debugger from trace data buffer. I.e. The line having the frame number for which following condition is met, will be highlighted in Trace window

(Memory location read by debugger) & (Address Mask) =

(Search address) & (Address Mask)

For the trace search, configured as shown in the figure below,

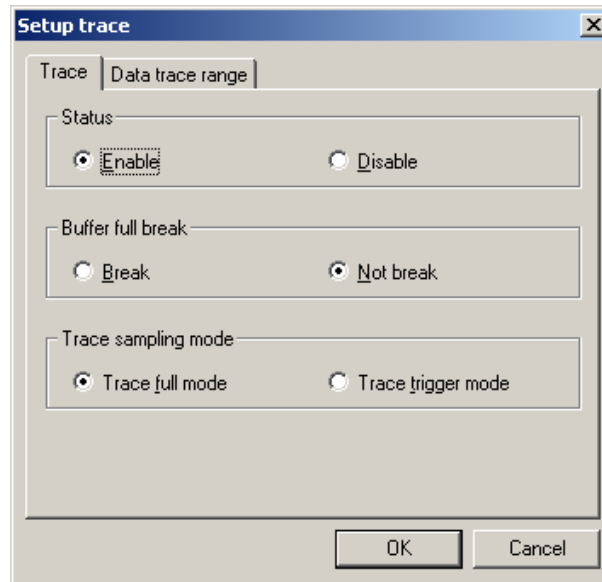


When every time *Find next* button is clicked, trace frame, containing memory address in the range of H'F800A0 to H'F800AF is highlighted. When search reaches the first frame number (-D'1) it rollbacks and search continues from last frame (-D'65535).

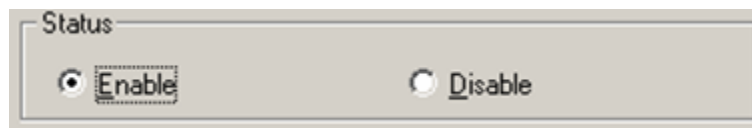
## 7.6 Trace Setup

Trace functionality can be configured by using *Setup Trace* dialog box.

Right click on *trace* window, on pop up menu, click on *Setup...*

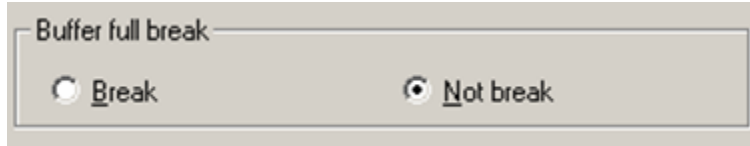


Trace data buffering can be enabled or disabled by selecting *Enable* or *Disable* radio button, under *Status* option.

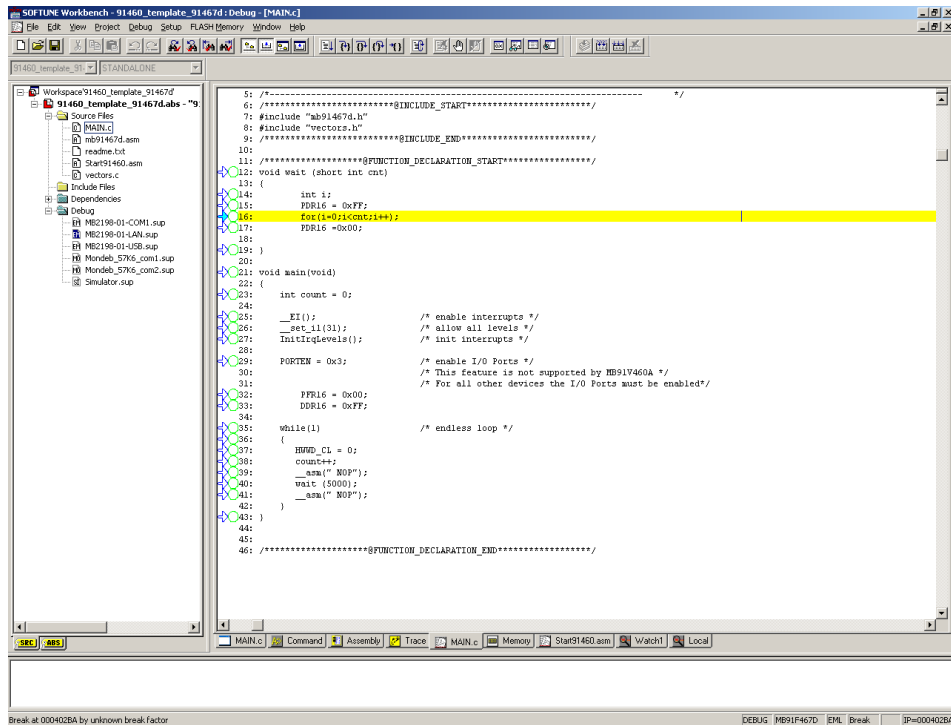


Trace data is buffered in the trace buffer. The trace buffer becomes full some time during debugging because its size is finite. When the trace buffer becomes full, the program being executed can be stopped. Trace buffer full break is set by selecting **Break** radio button under **Buffer full break** option.

MB2198-01 Emulator has 64K frames internal trace buffer and 256M frames External trace buffer.

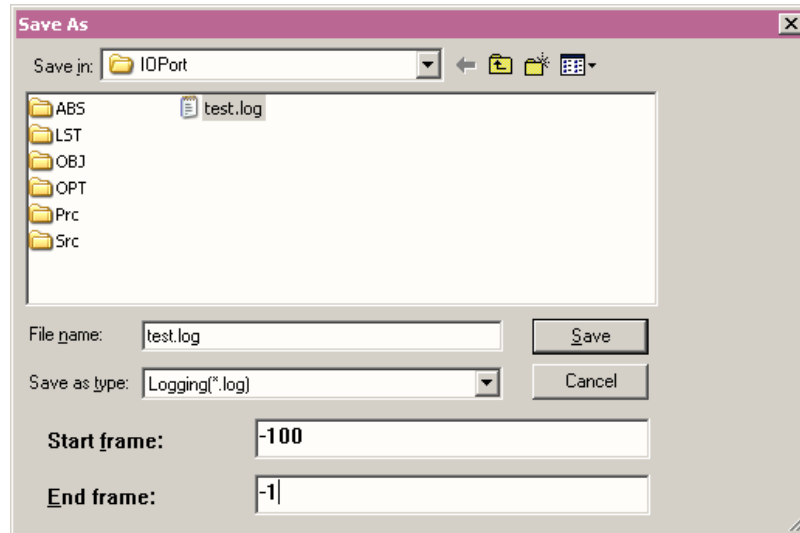


When program is halted by trace data buffer full break, corresponding line will be highlighted with yellow background.



## 7.7 Saving Trace Data

Since Softune version V30L34R05 it is possible to save the trace data beginning with a starting frame and ending with an end frame. Please enter negative numbers due to negative trace frames.



The save dialog is available by right-clicking in the trace window and choosing *Save file ...*.

# 8. Time Measurement



## How to Use Time Measurement Function

The Time Measurement is the simplest method of measurement. A 64-bit counter is used for this measurement resulting maximum cycle count of 18446744073709551615.

In this method two breakpoints have to be set. After the program has run from breakpoint 1 to 2 the Time Measurement will show the difference between the breakpoints.

The screenshot shows a dialog box titled "Measurement time" with the following data:

- From Initialize: 0h00m00s452ms986us725ns[Time] (Callout: Time from Reset to 1<sup>st</sup> Breakpoint)
- From Last Executed: 0h00m00s152ms198us075ns[Time] (Callout: Time from 1<sup>st</sup> Breakpoint to 2<sup>nd</sup> Breakpoint)
- From Initialize: 929982[Cycle] (Callout: Number of cycles from Reset to 1<sup>st</sup> Breakpoint)
- (929981 - 929999)
- From Last Executed: 328302[Cycle] (Callout: Number of cycles from 1<sup>st</sup> Breakpoint to 2<sup>nd</sup> Breakpoint)
- (328301 - 328319)

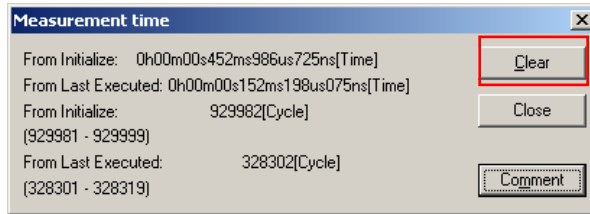
Buttons: Clear, Close, Comment

Now, take a look at example as shown below, in a *main()* function, two breakpoints are set. In a *wait()* function *PORT00* is set to high when entered in function and then set to low when left the function.

```
23: void main(void)
24: {
25:
26:     DDR00 = 0xFF;
27:     __asm(" NOP");
28:     wait (5000);
29:     __asm(" NOP");
30:
31: }
32:
33:
34:
35: void wait (int cnt)
36: {
37:     int i ;
38:     PDR00 = 0xFF;
39:     for (i = 0; i< cnt ; i++);
40:     PDR00 = 0x00;
41:
42:
43: }
```

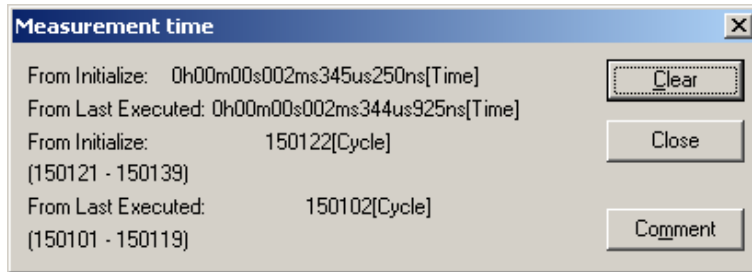


On Debug pull down menu click on *Time Measurement...*, Click on Clear if there are any time entries not equal to zero, Click on Close then.



Run the program. The MCU will stop at the beginning of the main function (1st breakpoint). This first stop initializes the time measurement counter. Next, run the program till the second breakpoint.

Now on Debug pull down menu click on *Time Measurement...*, one can find the following information



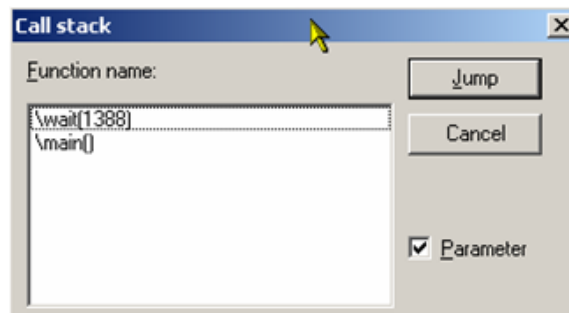
# 9. Call Stack



## How to Use Call Stack Feature

Usually, a program is a set of several subroutines. For this reason, as debugging advances, function calls of several stages occur. For example, one routine calls another and the called routine further calls another.

Call stack window can be opened by clicking on *Stack* under *Debug* menu



The call stack retains the relationship between function calls. Clicking a function name from the function name list immediately displays information for the function in the Source Window.

The function written in the lowermost line of the function name list is the main function. This main function calls the function above it. The called function further calls a function above it. In this way, the function written in the uppermost line is the function in which the current PC exists. When return is executed, functions are deleted in turn from the function name list, starting from the uppermost line.

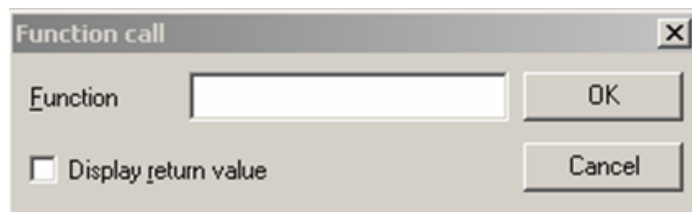
When a check mark is set to *Parameter*, an argument value is displayed after each function name, as shown in above figure. When no check mark is set, only parentheses "(" )" are displayed after each function name.

# 10. Function Call

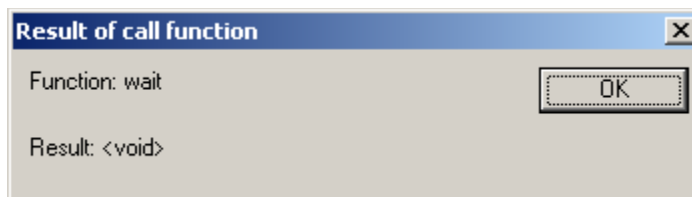


## How to Use Function Call Feature

The specified function can be started during debugging without reference to the flow of the program. This function is known as function call. This feature can be used when execution is stopped. After MCU reset one should at least execute one instruction before using this feature. The *Function call* dialog box is opened by right clicking on *Call...* on Debug pull down menu



When the function call dialog box opens, specify the function you want to call with a correct argument. If a breakpoint is set in the called function, the program stops at this breakpoint. When processing of the called function is terminated and control is returned, the function call result dialog box opens, if *Display return value check box* is checked. The PC then returns to the value before the function was called.



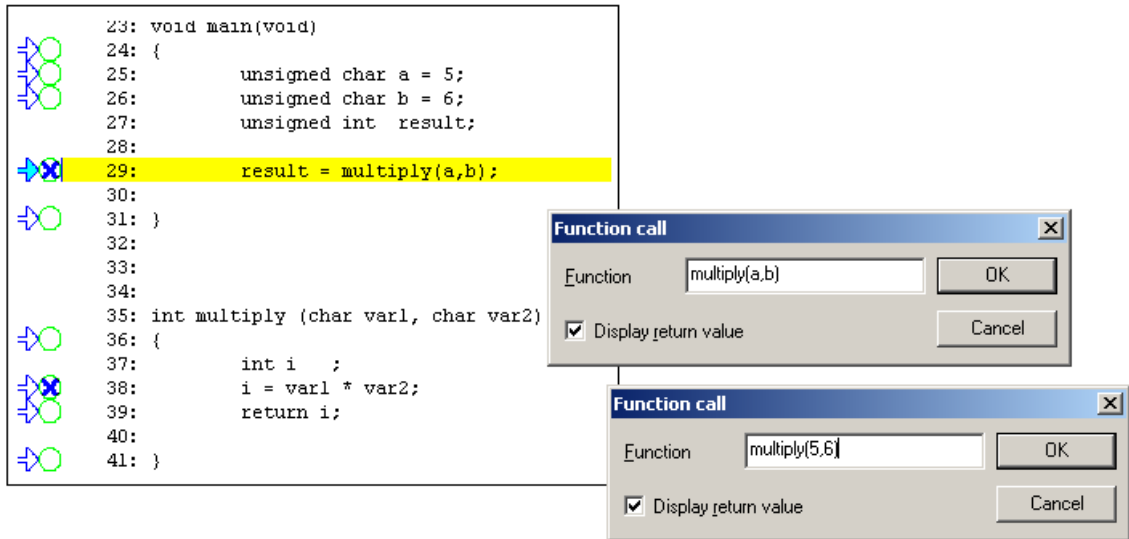
To understand the function, let us consider following example code

```
23: void main(void)
24: {
25:     unsigned char a = 5;
26:     unsigned char b = 6;
27:     unsigned int result;
28:
29:     result = multiply(a,b);
30:
31: }
32:
33:
34:
35: int multiply (char var1, char var2)
36: {
37:     int i ;
38:     i = var1 * var2;
39:     return i;
40:
41: }
```

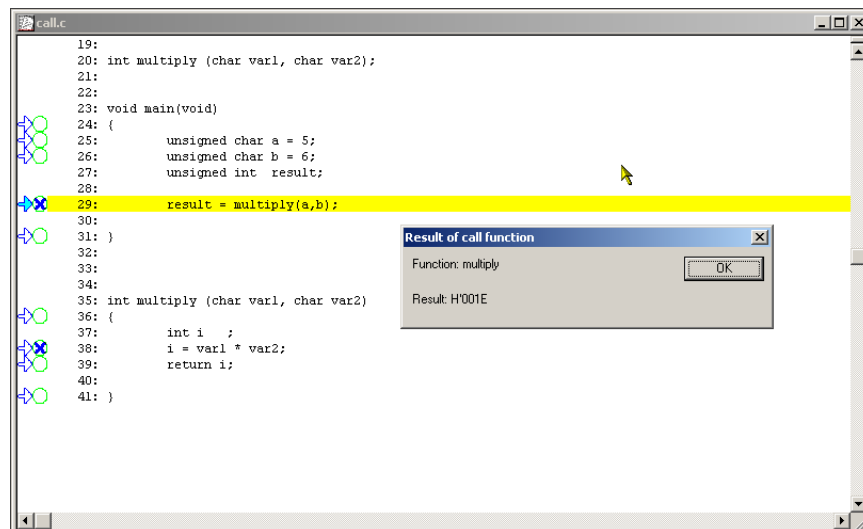
Let us set break point at line no. 29 and line no. 38. Start executing program, at first it halts at first break point. Now open *Function Call* dialog box (Debug -> Call...)

When the function definition is, *int multiply (char var1, char var2)*, specify the function call as follow

- multiply (5, 6); Where a constant value is directly specified
- multiply (a, b); Where variable 'a' & 'b' is directly specified



Since break point is set in the called function, the program stops at this breakpoint. When run again at the end of processing of called function, control is returned, the *Result of call function* dialog box as shown below opens. The PC then returns to the value before the function was called.



# 11. Vector



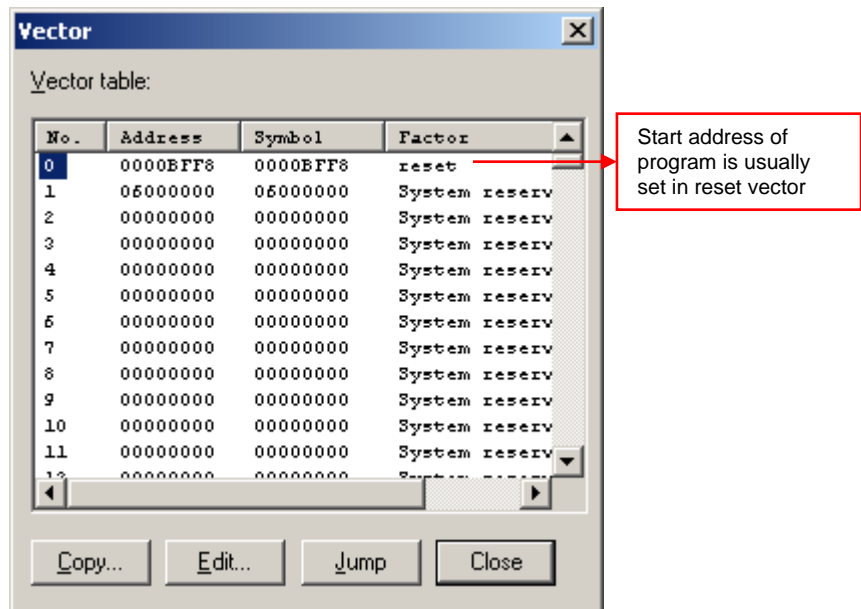
## How to Use Vector Feature

An interrupt vector is the memory address of an interrupt handler, or an index into an array called an interrupt vector table or dispatch table. Interrupt vector tables contain the memory addresses of interrupt handlers. When an interrupt is generated, the processor saves its execution state via a context switch, and begins execution of the interrupt handler at the interrupt vector.

## 11.1 Display and setting vectors

### 11.1.1 Display

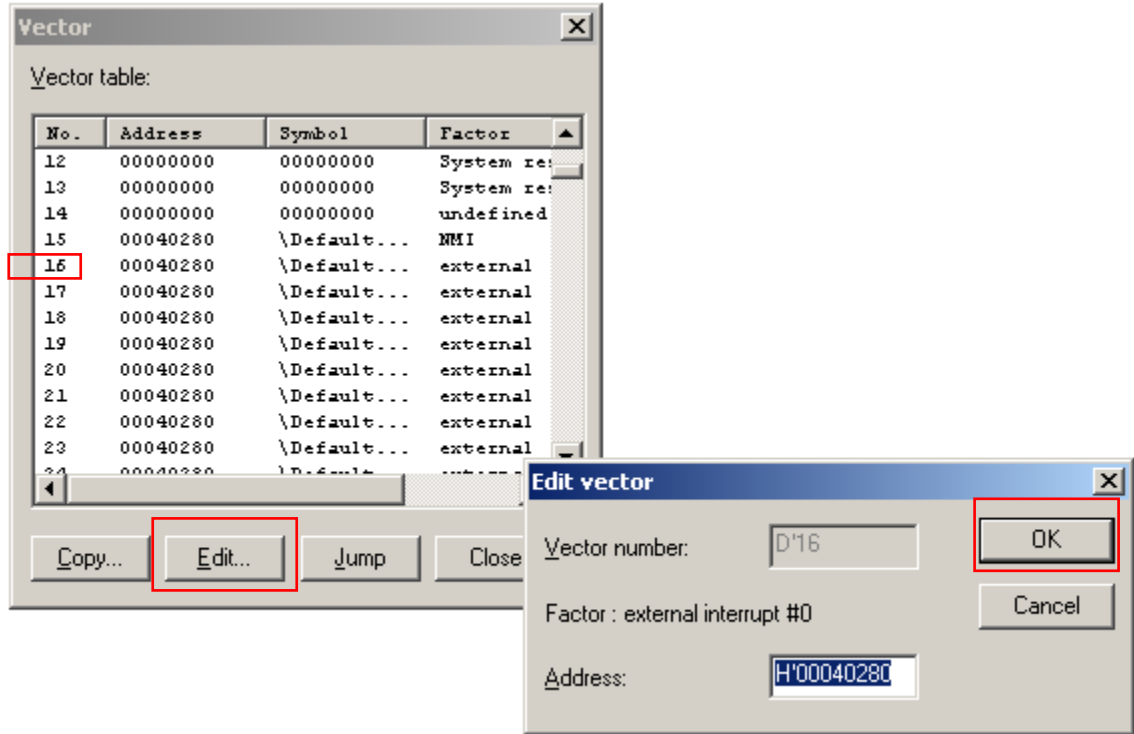
To display Interrupt vector table, click on *Vector...* at *Debug* pull down menu



### 11.1.2 Setting an address

Change the address set in a vector in the following procedure:

Select a vector table number and then click the *Edit* button. The vector edit dialog box shown below opens.



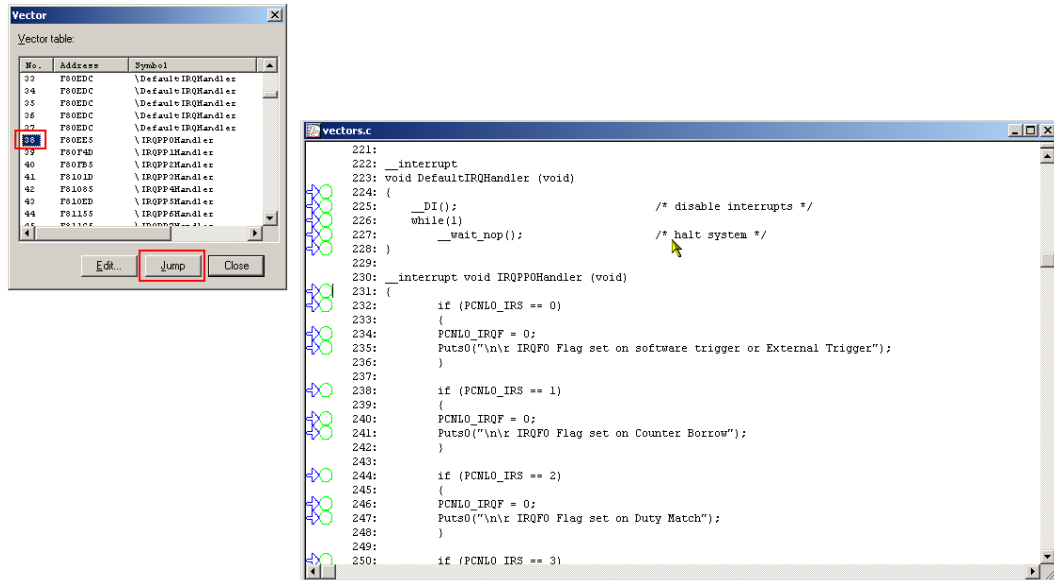
Set an address and then click the [OK] button.

## 11.2 Jump

Display the source of the program stored at the address set in the vector table in the following procedure:

1. Select a vector number.

Click the [Jump] button.



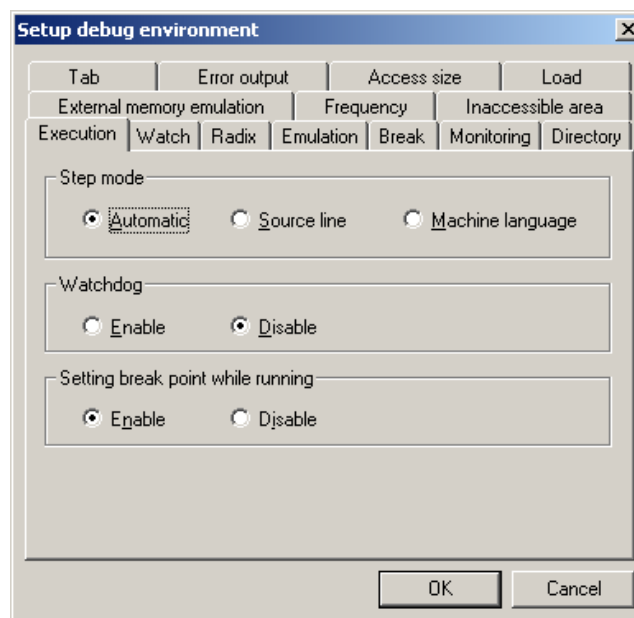
If the starting address of the program set in the vector table is incorrect, the source cannot be displayed (disassemble display).

The jump function merely displays the jump destination program; it does not update the program counter to move control to the address set in the vector table.

# 12. Debug Environment Setup Procedure



## 12.1 Execution



### Step mode

- Automatic  
Automatically sets the step unit according to the window display state. If it is a source code view, step will pass through each source line no. If it is a Mix mode code view, step will pass through disassemble source line no.
- Source Line  
Executes the step in units of source lines.
- Machine Language  
Executes the step in units of machine languages.

### Watchdog

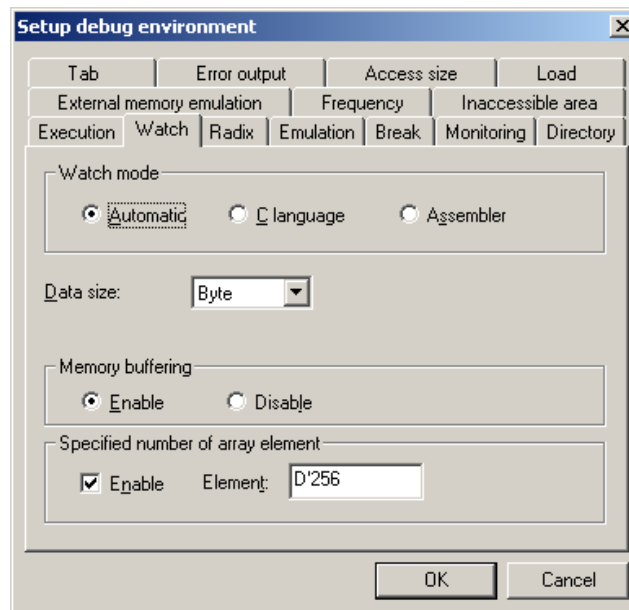
Specify whether to enable or disable the watchdog timer at program execution.

### Setting break point while running

If "Setting break point while running" is enabled, it is also possible to break settings even when executing a user program.



## 12.2 Watch



### Watch Mode

- Automatic  
Sets the watch mode automatically according to the analysis result. Refer 4.4 & 4.3
- C Language  
Sets the C language mode (interpretation as C language expressions).  
Refer 4.4 & 4.3
- Assembler  
Sets the assembler mode (interpretation as assembler expressions).  
Refer 4.4 & 4.3

### Data Size

Sets the display size in the assembler mode to either byte, word, long, single or double

### Memory Buffering

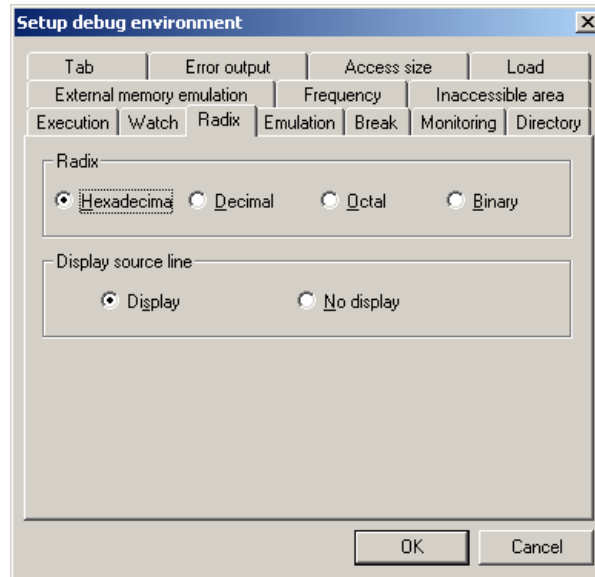
- Enable  
In case of variables as arrays or structures, memory of whole variables is read.  
They are accessed by size of the top variable.
- Disable  
In case of variables as arrays or structures, the memory of each variable is read.

### Specified number of array element

- Enable  
Debugger displays a warning dialogue in case of bigger array element than the number of array-element that you limited, when you register or expand an array with a watch variable.

- element  
 One can specify number (a default is D'256) of array element.  
 The default of this control is "Enable".

## 12.3 Radix



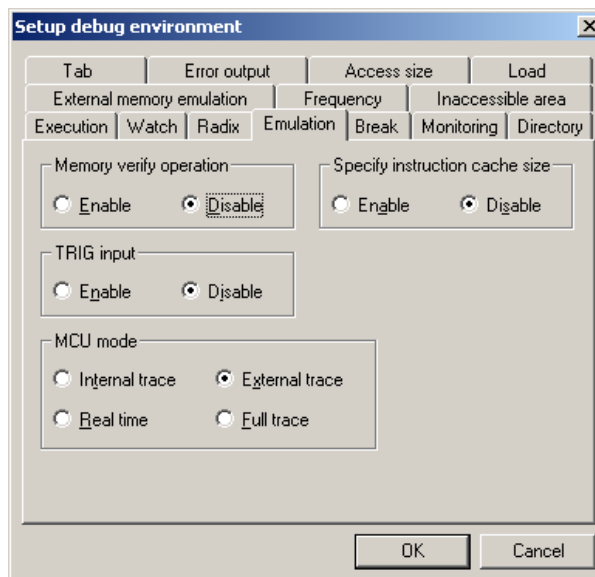
### Radix

Sets the default base number for numerical value.

### Display Source Line

Switches source line display to source line non display or vice versa for Trace- Instruction view.

## 12.4 Emulation



**Memory Verify Operation**

Specifies whether to verify memory when data is written to memory.

**TRG Input**

Specifies whether to enable or disable TRIG pin input. For more information please refer MB2198\_01 Hardware manual (MB2198-01-CM71-00413-2E).

**MCU Mode**

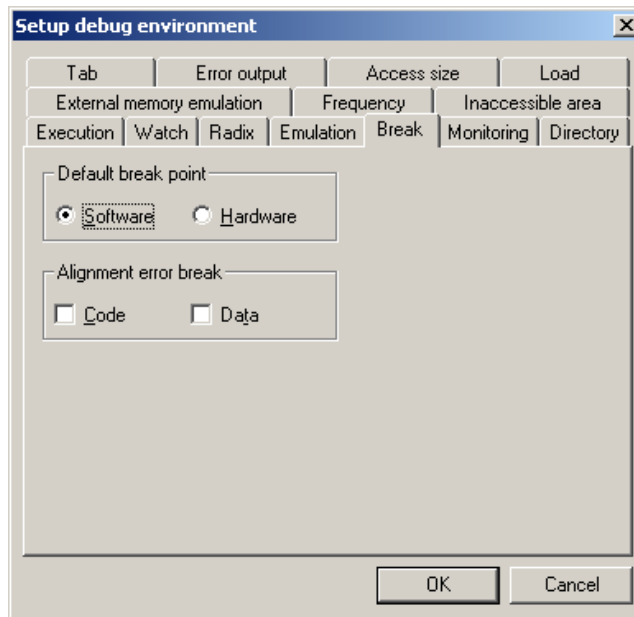
Specifies an MCU operation mode (internal trace, external trace, full trace mode or real-time mode).

**Specify instruction cache size**

Sets whether to automatically flush instruction cache.

## 12.5 Breakpoint

During program execution a break point can be set if the following set up is done:



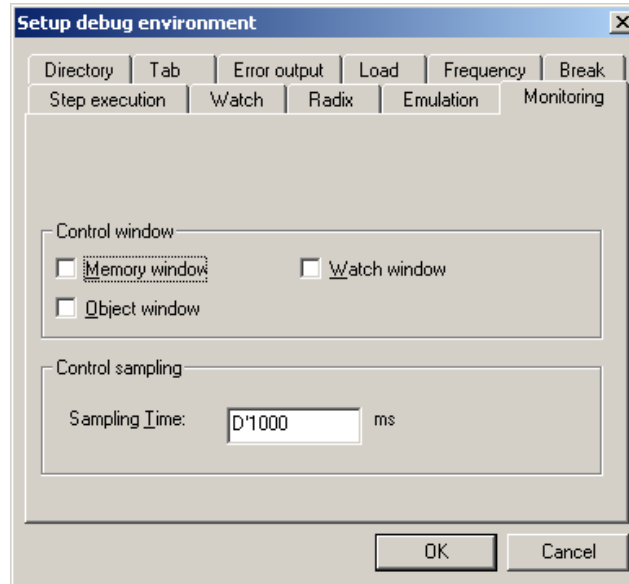
**Default break point**

Specifies the default type of the code breakpoint.

**Alignment error break**

Specifies whether to suspend MCU execution when an alignment error occurs

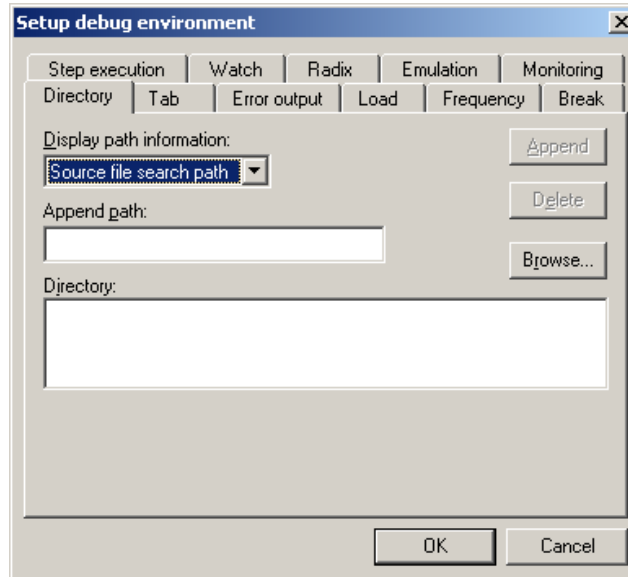
## 12.6 Monitoring



### Control Window

- Memory Window  
Specifies whether to monitor the Memory Window.
- Watch Window  
Specifies whether to monitor the Watch Window.
- Object Window  
Specifies whether to monitor the Object Window.
- Control Sampling  
Specifies sampling time for Watch window, Memory window and Object window

## 12.7 Directory



### Display path information

Specifies the path information to be displayed.

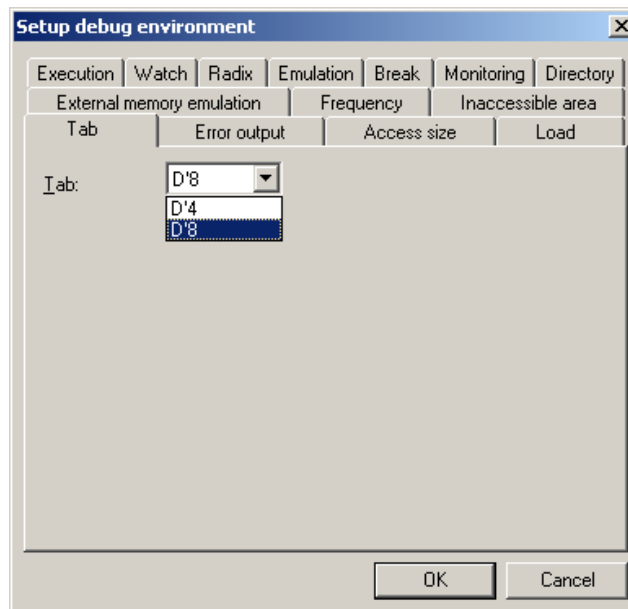
### Append path

Sets the path to be added.

### Directory

Displays the currently set directory.

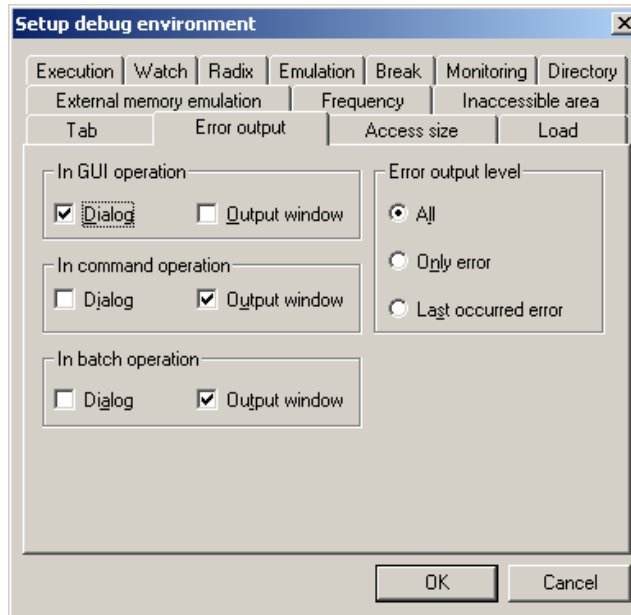
## 12.8 Tab



**Tab**

Specifies the Tab. (D'4/D'8)

## 12.9 Error output



**In GUI Operation**

Specifies where to output an error at GUI operation.

**In Command Operation**

Specifies where to output an error at command operation.

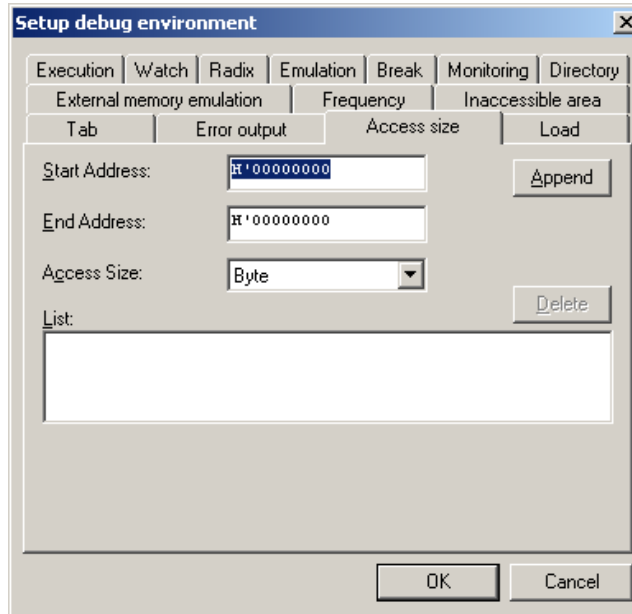
**In Batch Operation**

Specifies where to output an error at batch operation.

**Error Output Level**

Sets the output type when several errors occur.

## 12.10 Access Size



It is a function to set access size when the debugger accesses memory. When this setting is not done, the debugger does memory access by a command qualifiers or the most suitable size. Because it is set automatically about a built-in resource, setting is unnecessary by this function. However, the debugger does memory access by byte size on FILL, MOVE, COMPARE commands.

### **Start Address**

Specifies the start address to be set.

### **End Address**

Specifies the end address to be set.

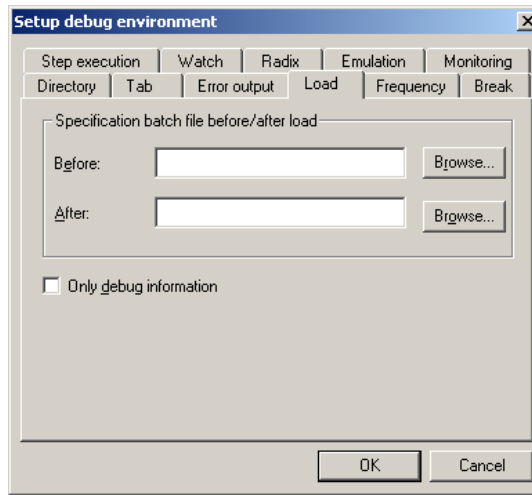
### **Access Size**

Specifies the access size to be set. (Byte/Halfword/Word)

### **List**

Displays the currently set area.

## 12.11 Load



This sets the environment when loading a target file registered in the project.

### Specification Batch File before/after load

- Before

This specifies the batch file to execute prior to the loading of the target file. This can also be changed using the Debugger's setup wizard.

- After

This specifies the batch file to execute after the loading of the target file. This can also be changed using the debugger's setup wizard.

- Debug Information Only

This specifies whether or not to load debug information. When checked, only the debug information is loaded.

Consider a system where MCU application program is stored in external flash and MCU is to be started in external vector fetch mode.

For the emulation of such a system one can connect the target with this check box marked. By doing this only debug information is loaded and with the help of Softune one can debug the application program. The restriction in this scenario is one can use only hardware breakpoint for debugging.

- Auto Mapping

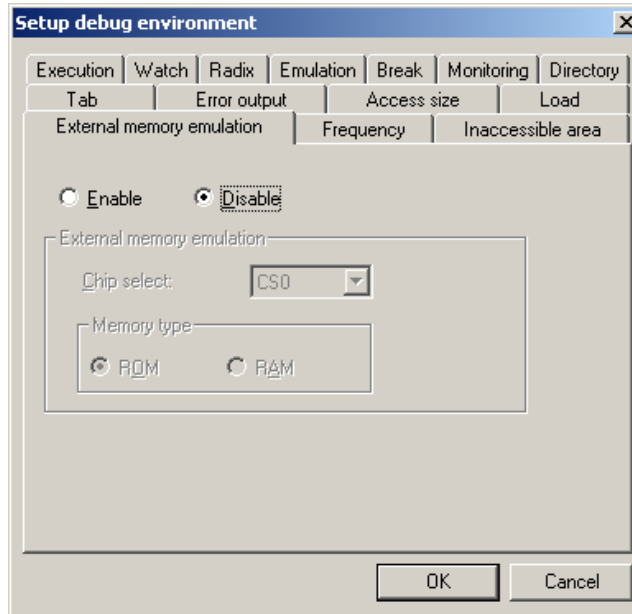
This specifies whether or not to enable the auto-map setting. When checked, auto-map setting is enabled.

- On demand load

Set whether to on demand load debug information. When a checkmark is placed in the checkbox, debug information is on demand loaded.



## 12.12 External Memory Emulation



### Enable/disable

Whether to enable or disable the external memory emulation function is specified.

### Chip select

The chip select number that can be output to the external bus is specified.

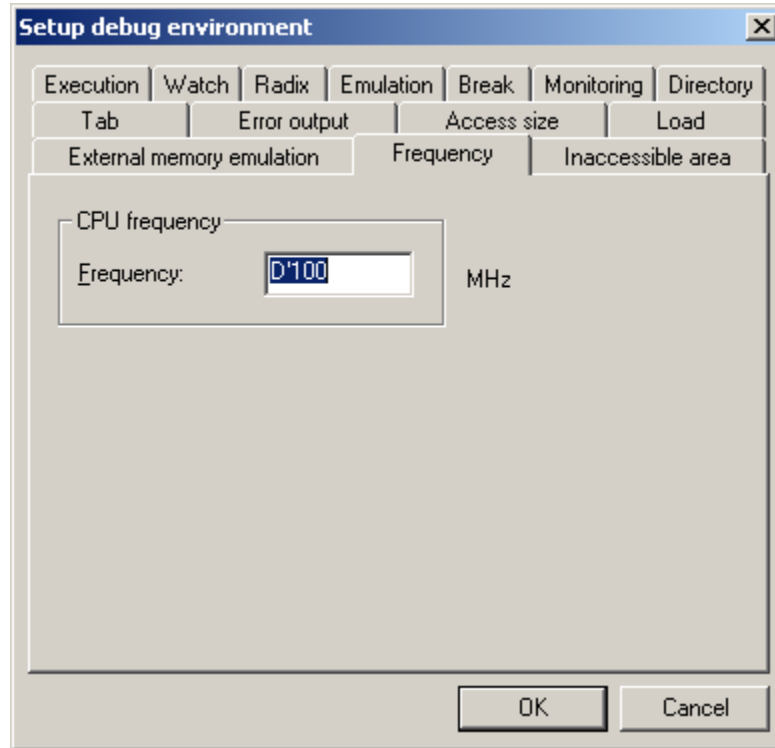
FR system: CS1 to CS5

FRex system: CS0 to CS7

### Memory type

Whether to allow or inhibit write access to external memory is specified.

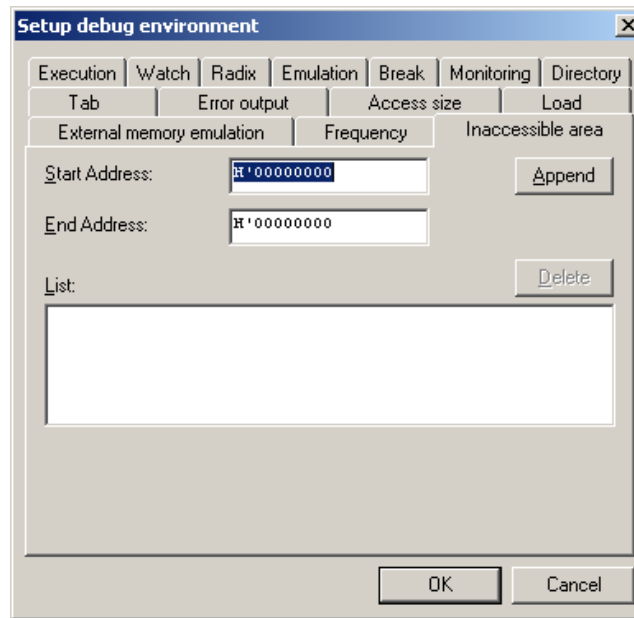
## 12.13 Frequency



### CPU Frequency

Set maximum CPU frequency. (Lower value results in higher responsiveness of emulation system.)

## 12.14 Inaccessible area



Selects default code break point (i.e. Software or Hardware)

This function inhibits access to debugger memory. Up to 16 areas can be set (by increments of one byte).

**Start Address**

Specifies the start address to be set.

**End Address**

Specifies the end address to be set.

**List**

Displays an regions being currently set. When the check mark of the area is removed, that the area is invalidated.

# 13. Trigger-Input and Emulator-Output



How to Use the Two BNC Connectors

## 13.1 The BNC Connectors



The MB2198-01 emulator has two BNC connectors. The left one is the “TRIG”-Input and the right one the “EMUL”-Output.

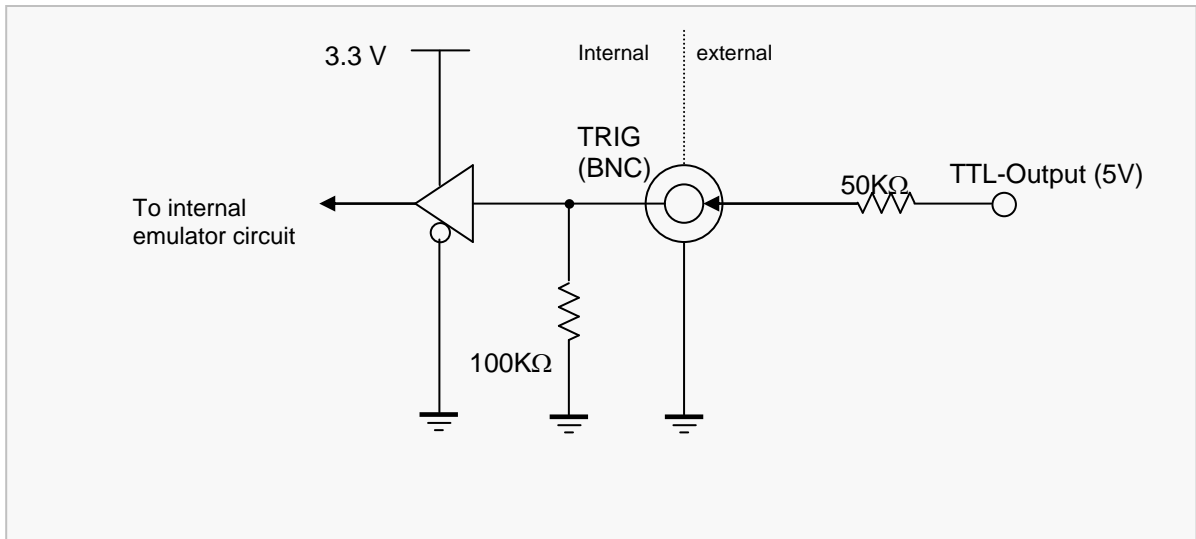
## 13.2 Trigger-Input

With this input an execution can be stopped. The Trigger-Input is a hardware “break point”.

A logical “high” (= 3.3V) on this input stops the execution in the debugging mode. Note, that because of internal latches and different clock speeds of the emulator and the MCU the termination is not immediately. The break slip is in a range of dozens to hundreds machine clock cycles.

The execution can be resumed after a triggered break.

Because of the 3.3V input and an internal 100K pull down resistor, it is recommended to use a serial 50K resistor, if a 5V signal is used:



### 13.3 Emulator-Output

The BNC-Output “EMUL” goes logical “high” (= 3.3V) if a program is executed and is “low” (= 0V) if the program is stopped or a break point has occurred. This signal can be used for controlling external hardware.

# 14. Installing LAN



This Chapter Describes How to Install the LAN Interface

## 14.1 Overview

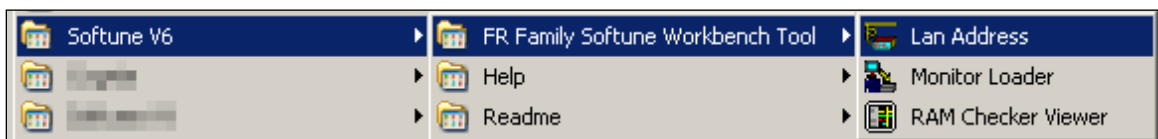
The emulator MB2198-01 are provided with a local area network adapter that can be used to program and debug a device over a network connection. No additional hardware connection except the LAN connection itself is required for this purpose

Using the Cypress LAN-remote controlled debug facility,

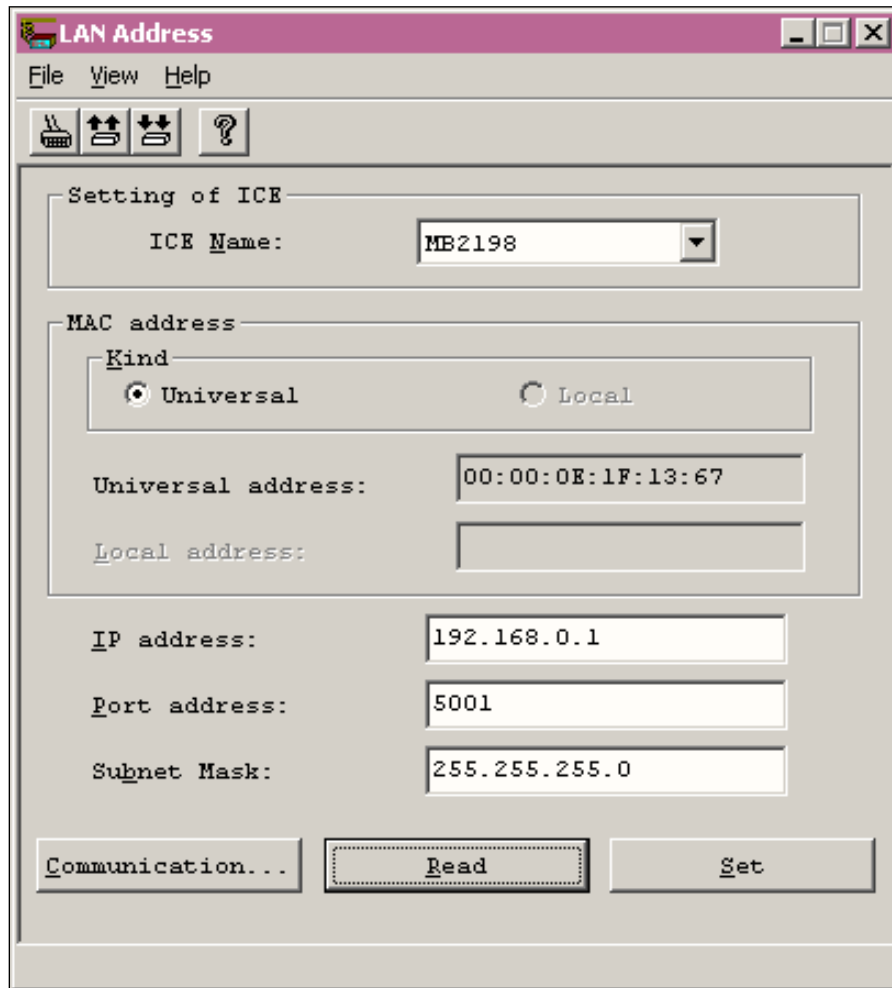
- A Cypress support engineer can easily help solving a concrete problem by debugging your application out of a Cypress support centre.
- You can control the emulator from different locations without having to move your hardware installation from one place to another.
- Program download will be more than 6 times faster than with RS232C

## 14.2 Configuring the LAN Adapter

1. Refer to the LAN Installation Manual coming with the LAN Adapter. This Application Note will give only some additional information.
2. Connect emulator and PC by the serial RS232-Port (or USB-Port)
3. Start the „LAN Address“ Program in the Softune Workbench Folder



4. Click on *Communication* to define the serial RS232 Port where the emulator is connected to or use USB connection.
5. Read current status from the emulator
6. Set the unique IP address given from your network manager. This is an very important point, every IP-address within a network has to be unique.
7. Check the Port address: must be 5001
8. Make settings valid by „Set“, and reset the emulator when prompted
9. Close Program „LAN Address“



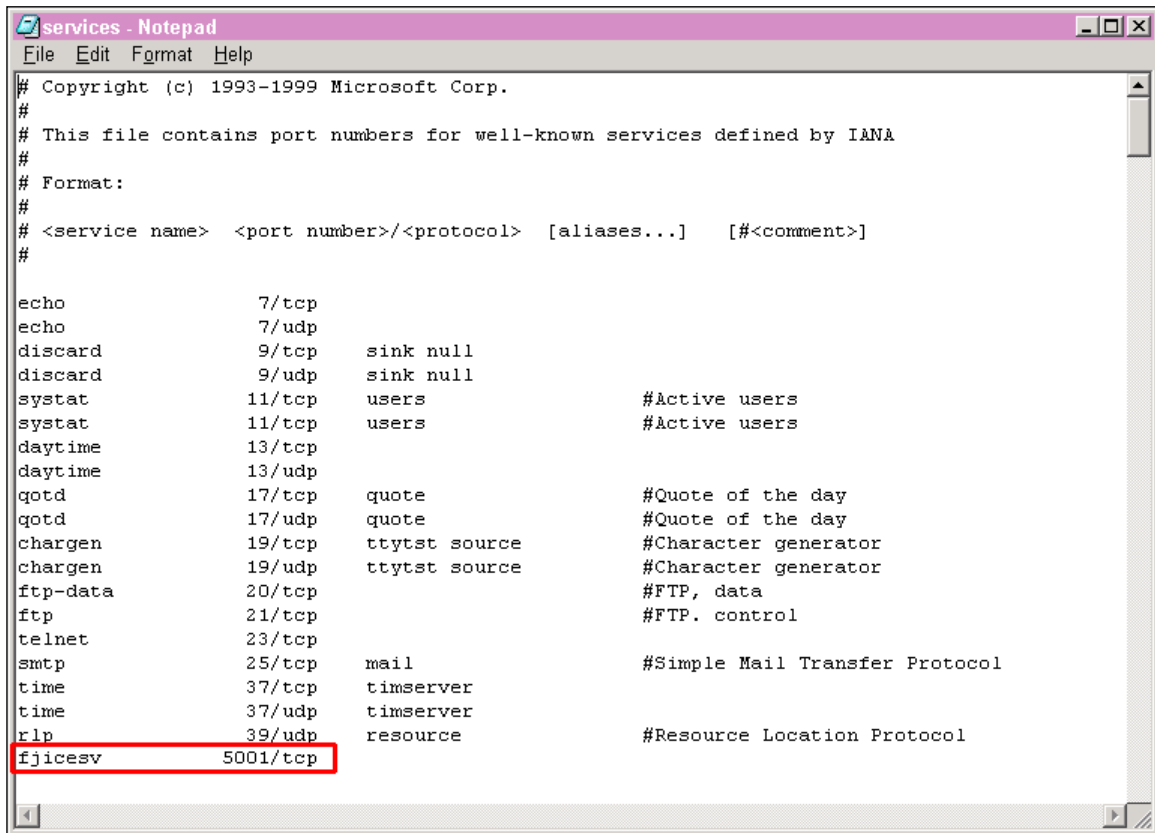
### 14.3 Configuring Operating System “Windows™,”

1. Within your Windows™-directory (e.g. C:\windows or C:\WINNT\system32\drivers\etc) you should find three files:
  - Services
  - Hosts
  - lmhosts[.sam]

(Note: The files may not have a file-extension!)
2. Make a copy off all three files, like for an example: services to sevices.old , etc.

Edit file services and add the following line:

```
fjicesv      5001/tcp          # Cypress emulator
```



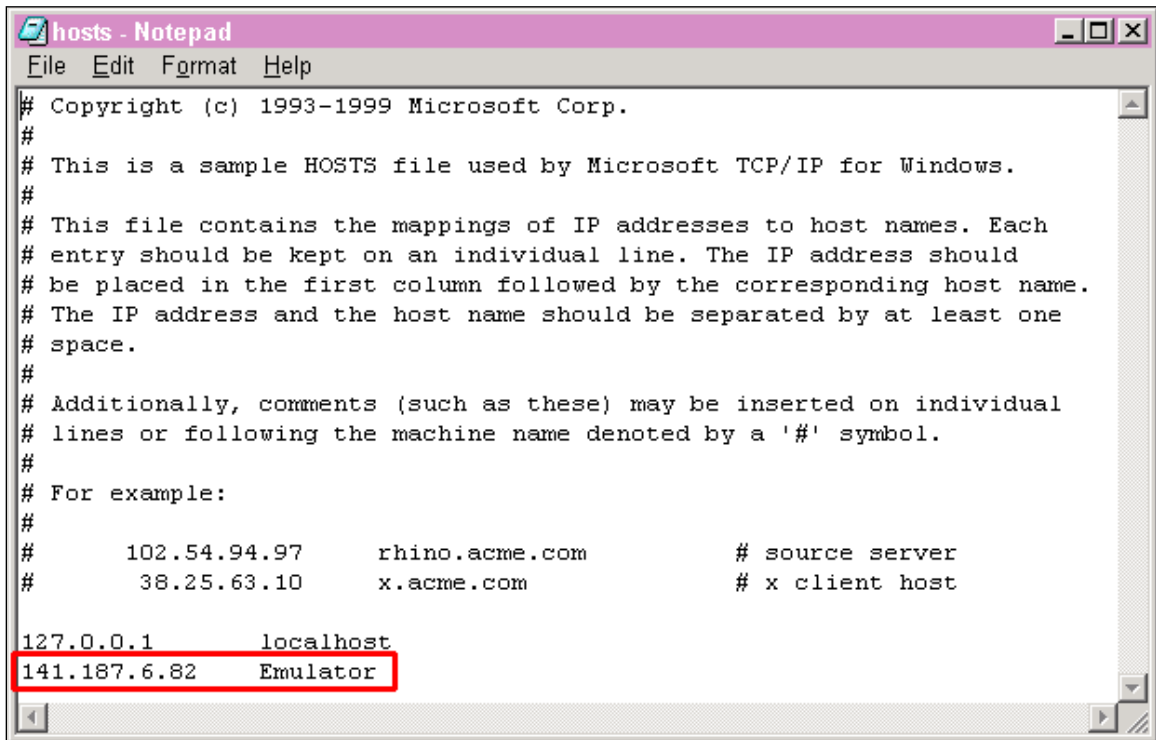
```

# Copyright (c) 1993-1999 Microsoft Corp.
#
# This file contains port numbers for well-known services defined by IANA
#
# Format:
#
# <service name> <port number>/<protocol> [aliases...] [#<comment>]
#
echo          7/tcp
echo          7/udp
discard      9/tcp      sink null
discard      9/udp      sink null
sysstat      11/tcp      users          #Active users
sysstat      11/tcp      users          #Active users
daytime      13/tcp
daytime      13/udp
qotd         17/tcp      quote         #Quote of the day
qotd         17/udp      quote         #Quote of the day
chargen      19/tcp      ttytst source #Character generator
chargen      19/udp      ttytst source #Character generator
ftp-data     20/tcp
ftp          21/tcp      #FTP. control
telnet       23/tcp
smtp         25/tcp      mail          #Simple Mail Transfer Protocol
time         37/tcp      timserver
time         37/udp      timserver
rlp          39/udp      resource      #Resource Location Protocol
fjicesv      5001/tcp
  
```

(note: if 5001/tcp is already contained in the file *services*, use an unused number beginning with 5002 or greater, e.g. *fjicesv 5002/tcp*. In that case also the emulator address given by the program „LAN address“ (see above) has to be changed!

3. When saving again the file *Services* be sure that your editor (e.g. notepad) will not add any extension, e.g. *.txt*, to your file. To get sure, use quotation marks for the filename:  
File save as: *"services"*
  
4. The file *hosts* is used to make a redefinition of the complex IP-number with a simple name within your global network. This may be important if you use a DNS-Server. Of course, you can define different names for the same IP-Address, as shown below. Edit file *hosts* and add the following line:  
 "The unique IP address (as set by LAN-Address, see above)" "Nickname of emulator"  
 e.g. 141.187.6.82    Emulator





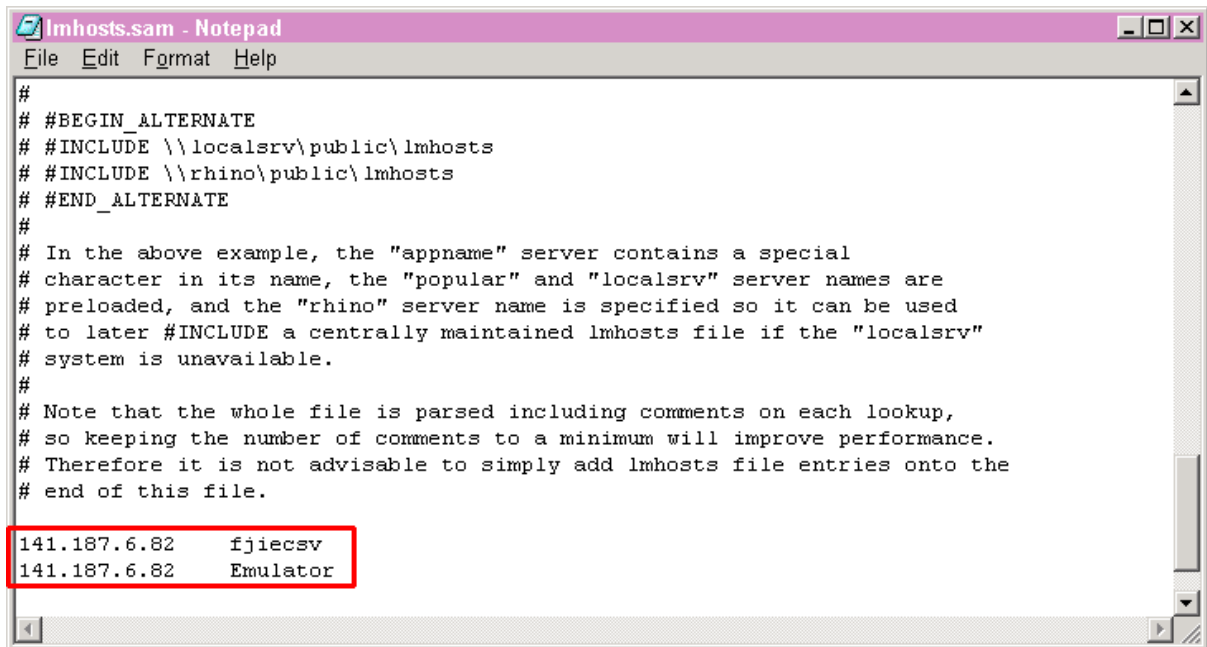
```

# Copyright (c) 1993-1999 Microsoft Corp.
#
# This is a sample HOSTS file used by Microsoft TCP/IP for Windows.
#
# This file contains the mappings of IP addresses to host names. Each
# entry should be kept on an individual line. The IP address should
# be placed in the first column followed by the corresponding host name.
# The IP address and the host name should be separated by at least one
# space.
#
# Additionally, comments (such as these) may be inserted on individual
# lines or following the machine name denoted by a '#' symbol.
#
# For example:
#
#       102.54.94.97       rhino.acme.com           # source server
#       38.25.63.10      x.acme.com             # x client host

127.0.0.1        localhost
141.187.6.82    Emulator
  
```

When saving again the file *Hosts* be sure that your editor (e.g. notepad) will not add any extension, e.g. *.txt*, to your file. To get sure, use quotation marks for the filename:  
File save as: "*hosts*"

5. The file *lmhosts* (or *lmhosts.sam*) is used to make a redefinition of the complex IP-number with a simple name within your local network. Of course, you can define different names for the same IP-Address, as shown below. Edit file *lmhosts* and add the following line:



“the unique IP address (as set by LAN-Address, see above)”      “nickname of emulator”

e.g.  
 141.187.6.53      fjiescv  
 141.187.6.53      Emulator

When saving again the file *Lmhosts* get sure that your editor (e.g. notepad) will not add any extension, e.g. .txt, to your file. To get sure, use quotation marks for the filename:  
 e.g. File save as: "*lmhosts*" (or *lmhosts.sam*)

## 14.4 Checking the network-connection

Disconnect serial RS232 (or USB) cable from Emulator and try to find the emulator:

- Open DOS-Window (or open RUN (Ausführen) in the Start-Menu)
- The command `ping Emulator` should acknowledge with some time-values, that means the network is set up right.

The emulator with LAN-adapter is successfully integrated in the network environment and can be used by the Softune Workbench.

## 14.5 Troubleshooting

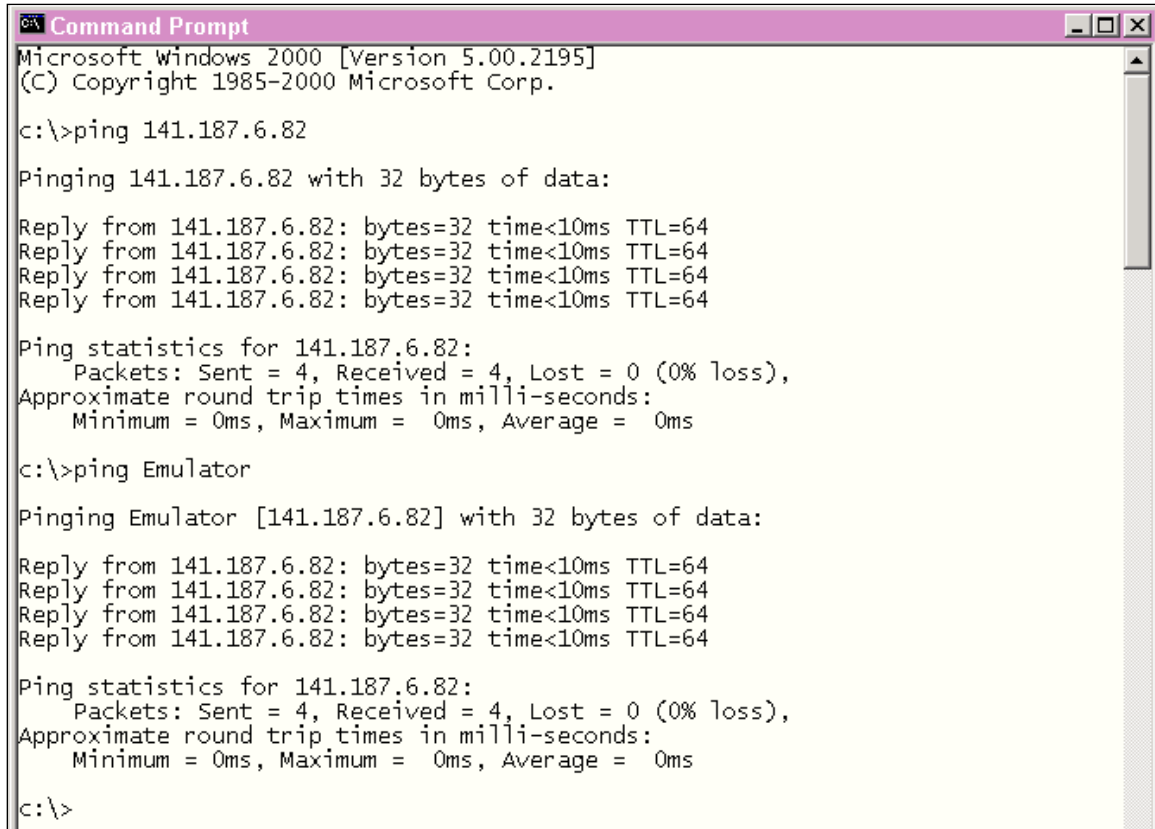
If the command `ping my_emulator` will reply with a timeout-message, try to find the emulator by its IP-address: Type command `ping IP-address`, e.g. `ping 141.187.6.53`.

If this will work, then check settings (nickname and IP-address) within *Lmhosts* and *Hosts*.

If neither nickname (`my_emulator`) nor IP-address will work, check settings done by the program *LAN-address* and check the file *Services*.

Also check your physical network interconnection cables. Please keep in mind, that the emulator only works with 10Mbit/s. This means in case that for a 100Mbit/s network a 100Mbits/10Mbit - HUB is needed. When using a HUB for 10Base-T then a standard (1:1) network cable has to be used. If the emulator is connected directly to the PC a „crossed network cable“ is necessary.

Good answer: Emulator/LAN-adapter replies with time-values:



```
Command Prompt
Microsoft Windows 2000 [Version 5.00.2195]
(C) Copyright 1985-2000 Microsoft Corp.

c:\>ping 141.187.6.82

Pinging 141.187.6.82 with 32 bytes of data:

Reply from 141.187.6.82: bytes=32 time<10ms TTL=64
Reply from 141.187.6.82: bytes=32 time<10ms TTL=64
Reply from 141.187.6.82: bytes=32 time<10ms TTL=64
Reply from 141.187.6.82: bytes=32 time<10ms TTL=64

Ping statistics for 141.187.6.82:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
    Approximate round trip times in milli-seconds:
        Minimum = 0ms, Maximum = 0ms, Average = 0ms

c:\>ping Emulator

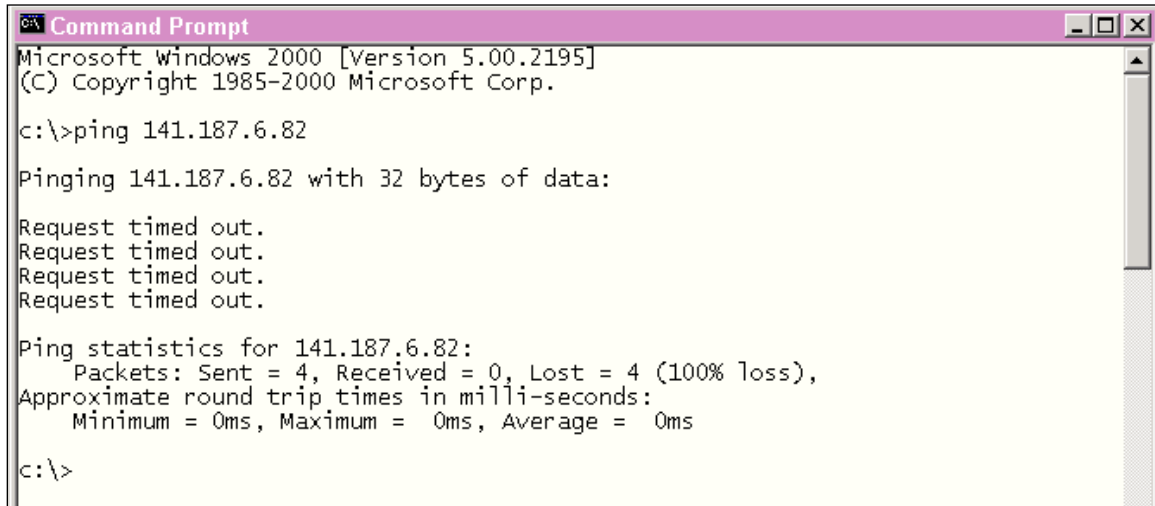
Pinging Emulator [141.187.6.82] with 32 bytes of data:

Reply from 141.187.6.82: bytes=32 time<10ms TTL=64
Reply from 141.187.6.82: bytes=32 time<10ms TTL=64
Reply from 141.187.6.82: bytes=32 time<10ms TTL=64
Reply from 141.187.6.82: bytes=32 time<10ms TTL=64

Ping statistics for 141.187.6.82:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
    Approximate round trip times in milli-seconds:
        Minimum = 0ms, Maximum = 0ms, Average = 0ms

c:\>
```

“Failed”-answer: Emulator/LAN-adapter replies with timeout message:



```
Command Prompt
Microsoft Windows 2000 [Version 5.00.2195]
(C) Copyright 1985-2000 Microsoft Corp.

c:\>ping 141.187.6.82

Pinging 141.187.6.82 with 32 bytes of data:

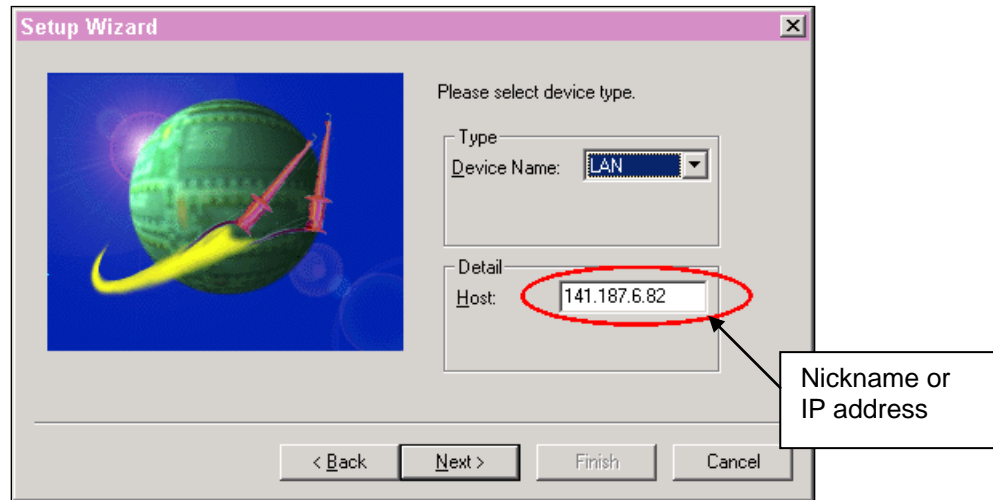
Request timed out.
Request timed out.
Request timed out.
Request timed out.

Ping statistics for 141.187.6.82:
    Packets: Sent = 4, Received = 0, Lost = 4 (100% loss),
    Approximate round trip times in milli-seconds:
        Minimum = 0ms, Maximum = 0ms, Average = 0ms

c:\>
```

## 14.6 Softune Workbench

Within the Softune Workbench the LAN interface can be used instead of a serial RS232C communication. Detail-Host: can be the „nickname“ as defined in the files *hosts* and *lmhosts* or the IP-address of the emulator, set by the program „LAN-Address“, can be used.



# 15. Miscellaneous



Remarks and Hints

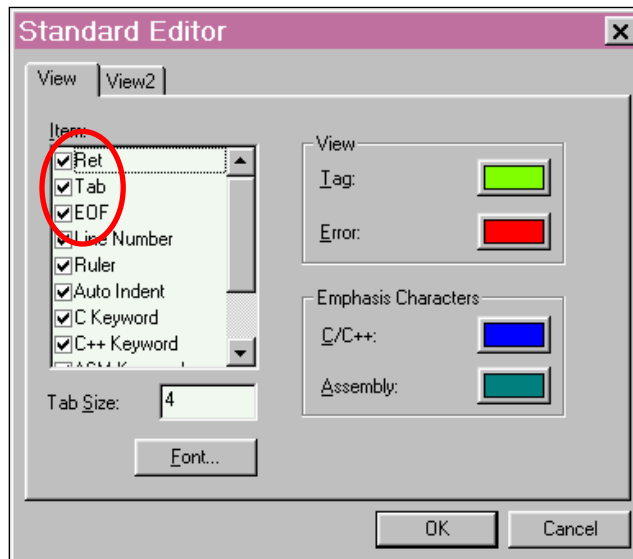
## 15.1 View Mode of the Editor

When starting the Softune Workbench Software for the first time, the text editor has some default viewing settings, which can be switched off. Source codes will look like in the left picture.

```
21 unsigned char LED;↓
22 ↓
23 /* Sub Routines */↓
24 ↓
25 void init_timer(void)↓
26 {↓
27 ^   TMRLR0 = 0x61A7;^ ^   /* relo
28 ^   TMCSR0 = 0x81B;^^ ^   /* pres
29 }↓
```

You can disable the viewing of the tabulators, Return signs and the End-of-File delimiter by clicking on the right mouse button just over the text window. Then a large pop up window will open.

Choose *Customize*. Then the following new window will occur:



The first three entries in the item list are for selecting and deselecting the view of these non-printing characters.

# 16. Appendix



Related Documents

## 16.1 Related Documents

Please find further information in the following documents.

- MB2198-01 Hardware Manual (Emulator)
- MB2198-01 Getting Started Application Note
- MB2198-01 Installation Guide Application Note

# 17. Additional Information



Information about Cypress Microcontrollers can be found on the following Internet page:

<http://www.cypress.com/cypress-microcontrollers>

# Revision History



## Document Revision History

Document Title: FR Family MB2198-01 Emulator System Getting Started Guide			
Document Number: 002-05222			
Revision	Issue Date	Origin of Change	Description of Change
**	11/13/2003	NOFL	Initial release
	01/25/2008		Installing LAN chapter added
	07/09/2008		Complete reorganisation and based on MB91460 Series
*A	03/22/2016	NOFL	Migrated Spansion Guide from MCU-AN-391027-E-V20 to Cypress format